MATLAB
开发实例系列图书



MATLAB 高效编程技巧与应用：25个案例分析

吴 鹏(rocwoods) 编著

北京航空航天大学出版社

策划编辑：陈守平

封面设计：

本书是作者八年MATLAB使用经验的总结，精心设计的所有案例均来自于国内各大MATLAB技术论坛网友的切身需求，其中不少案例涉及的内容和求解方法在国内现已出版的MATLAB书籍中鲜有介绍。

本书首先针对MATLAB新版本特有的一些编程思想、高效的编程方法、新技术进行了较为详细的讨论，在此基础上，以大量案例介绍了MATLAB在科学计算中的应用。内容包括：MATLAB快速入门、重新认识矢量（向量）化编程、MATLAB处理海量数据、匿名函数类型介绍、嵌套函数类型介绍、积分以及积分方程求解案例、优化及非线性方程（组）求解案例、人脸图像压缩与重建案例、有关预测分类的案例、常微分方程（组）求解案例、层次分析法及其MATLAB实现、定时器及其应用。

本书可作为高等院校本科生、研究生MATLAB课程的辅助读物，也可作为从事科学计算和算法研究的科研人员的参考用书。

上架建议：计算机语言与程序

ISBN 978-7-5124-0083-2



定价：39.00元

前　言

当今社会，数学作为一切学科基础地位的特征越来越明显，其重要性不言而喻。MATLAB从诞生那一天起，就为数学和实际应用之间架起了一座桥梁，如今经过20多年的发展，这座桥变得越来越雄伟、壮观。现如今，从国外高校到国内高校，从国外期刊到国内期刊，早已随处可见MATLAB应用的身影。更重要的是，近几年来，国内许多高新技术公司也开始普遍应用MATLAB来进行算法前期开发、验证。

如今，国内MATLAB相关书籍已经有很多，这些书籍极大地推动了MATLAB在国内的普及。但是MATLAB发展迅速，目前每年推出两个版本，现有书籍对MATLAB高版本一些特有的编程思想、高效的编程方法、新技术等，鲜有专门详细的讨论。

本书力图以一种全新的模式，从各个角度将MATLAB呈现给读者。全书共分两部分：第一部分（第1～5章）是有关MATLAB高效编程的一些方法、原则介绍；第二部分（第6～12章）是案例分析。关于高效编程，本书详细讨论了传统的向量化编程原则在新旧版本MATLAB下的异同，MATLAB如何处理海量数据，匿名函数和嵌套函数灵活、强大、富有弹性的功能。在案例分析部分，介绍了25个案例，这些案例都来自我平时的研究积累以及长期以来帮助网友解决的典型问题。案例涉及复杂的多重积分、积分方程、非线性方程求解、全局优化、遗传算法、Benders分解算法、人脸图像压缩与重建、灰色分析、距离判别法与Bayes判别法在分类中的应用、支持向量机、各类型的常微分方程（组）求解、层次分析法以及定时器的应用等。

互联网的兴起催生了很多专门讨论MATLAB相关技术的论坛、社区。这些社区集中了来自社会各行各业、高校各学科各专业的MATLAB使用者、爱好者。这些社区往往能够紧跟MATLAB的发展，并对其最新的技术及时作出反应。我从2005年开始就一直在国内一些成立较早的MATLAB论坛社区，如研学论坛、仿真科技论坛、振动论坛的MATLAB版面参与讨论问题，并发表了一系列技术精华帖。MATLAB中文论坛成立后，我通过该平台更是经常与各种程度的MATLAB使用者打交道，了解MATLAB使用者最容易遇到的一些问题，以及一些MATLAB软件最新的技术。可以将本书看成是我对这些经验的提取与总结。

我在上大学期间一度十分痛恨“数值计算”这门课程，因为为了应付考试而不得不背一些算法流程、公式并手动计算结果，这是非常枯燥和烦琐的。后来竟180度转弯，喜欢上了数值计算，这完全是因为MATLAB——转机就是大二下学期的数学建模，必须要借助MATLAB完成。当用MATLAB轻而易举地随意拟合了一个20多阶的多项式来近似一堆数据时，我被震撼了。这种震撼是忍受了长时间无比枯燥的手动计算后而发自肺腑的。当然现在看来，那时候的拟合毫无技术含量，毫无实际意义。但也正是因为这个开始，促使我不断去探索MATLAB。一开始没有计算机，就去看书，在图书馆里看各种有关MATLAB的书。随着看的书的增多，MATLAB在脑子里也越来越清晰了，对它的喜爱也越来越深。后来2005年在公司实习，查资料时偶然进入论坛这片新天地，蓦然发现居然有那么多相同爱好的人，从此一发不可收拾，便开始了和一帮志同道合的朋友切磋提高的过程。

针对本书，北京航空航天大学出版社和MATLAB中文论坛（http://www.ilovematlab.

cn/)特别提供了读者与作者在线交流的平台（http://www.ilovematlab.cn/forum-182-1.html），我希望借助这个平台实现与广大读者面对面交流，解决大家在阅读此书过程中遇到的问题，分享彼此的学习经验，从而达到共同进步。本书所有源程序以及测试数据将放到MATLAB中文论坛的读者与笔者在线交流平台及北京航空航天大学出版社的网站上，供读者自由下载。

特别感谢北京航空航天大学出版社陈守平编辑，在她一再支持与鼓励下，我决定接受挑战，完成本书创作。

在本书的写作过程中，我得到了天津科技大学谢中华老师、MATLAB中文论坛创始人math(张延亮)和仿真科技论坛前总版主bainhome(马良)的支持与鼓励，在此，向他们表示最真诚的谢意。

最后，还要感谢我的家人和朋友，在他们无微不至的关心与支持下，我顺利完成本书的写作，在此，向他们表示最衷心的感谢。

由于作者水平有限，书中难免出现错误，恳请广大读者和同行批评指正。本书勘误网址：http://www.ilovematlab.cn/thread-81886-1-1.html。

吴　鹏

2010年3月于北京市昌平区

内容简介

本书是作者八年 MATLAB 使用经验的总结，精心设计的所有案例均来自于国内各大MATLAB技术论坛网友的切身需求，其中不少案例涉及的内容和求解方法在国内现已出版的MATLAB书籍中鲜有介绍。

本书首先针对 MATLAB 新版本特有的一些编程思想、高效的编程方法、新技术进行了较为详细的讨论，在此基础上，以大量案例介绍了 MATLAB 在科学计算中的应用。内容包括：MATLAB快速入门、重新认识矢量(向量)化编程、MATLAB 处理海量数据、匿名函数类型介绍、嵌套函数类型介绍、积分以及积分方程求解案例、优化及非线性方程(组)求解案例、人脸图像压缩与重建案例、有关预测分类的案例、常微分方程(组)求解案例、层次分析法及其 MATLAB 实现、定时器及其应用。

本书可作为高等院校本科生、研究生 MATLAB 课程的辅助读物，也可作为从事科学计算和算法研究的科研人员的参考用书。

图书在版编目(CIP)数据

MATLAB 高效编程技巧与应用:25 个案例分析/吴鹏编著. --北京:北京航空航天大学出版社,2010. 6

ISBN 978-7-5124-0083-2

Ⅰ. ①M… Ⅱ. ①吴… Ⅲ. ①计算机辅助计算—软件包,MATLAB Ⅳ. ①TP391.75

中国版本图书馆 CIP 数据核字(2010)第 079425 号

MATLAB 高效编程技巧与应用:25 个案例分析

吴 鹏(rocwoods) 编著

责任编辑 陈守平

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191) http://www.buaapress.com.cn

发行部电话:(010)82317024 传真:(010)82328026

读者信箱:bhpress@263.net 邮购电话:(010)82316936

北京市媛明印刷厂印装 各地书店经销

*

开本:787 mm×1 092 mm 1/16 印张:17 字数:435 千字

2010 年 6 月第 1 版 2010 年 6 月第 1 次印刷 印数:4 000 册

ISBN 978-7-5124-0083-2 定价:39.00 元

目　　录

第一部分　高效编程技巧

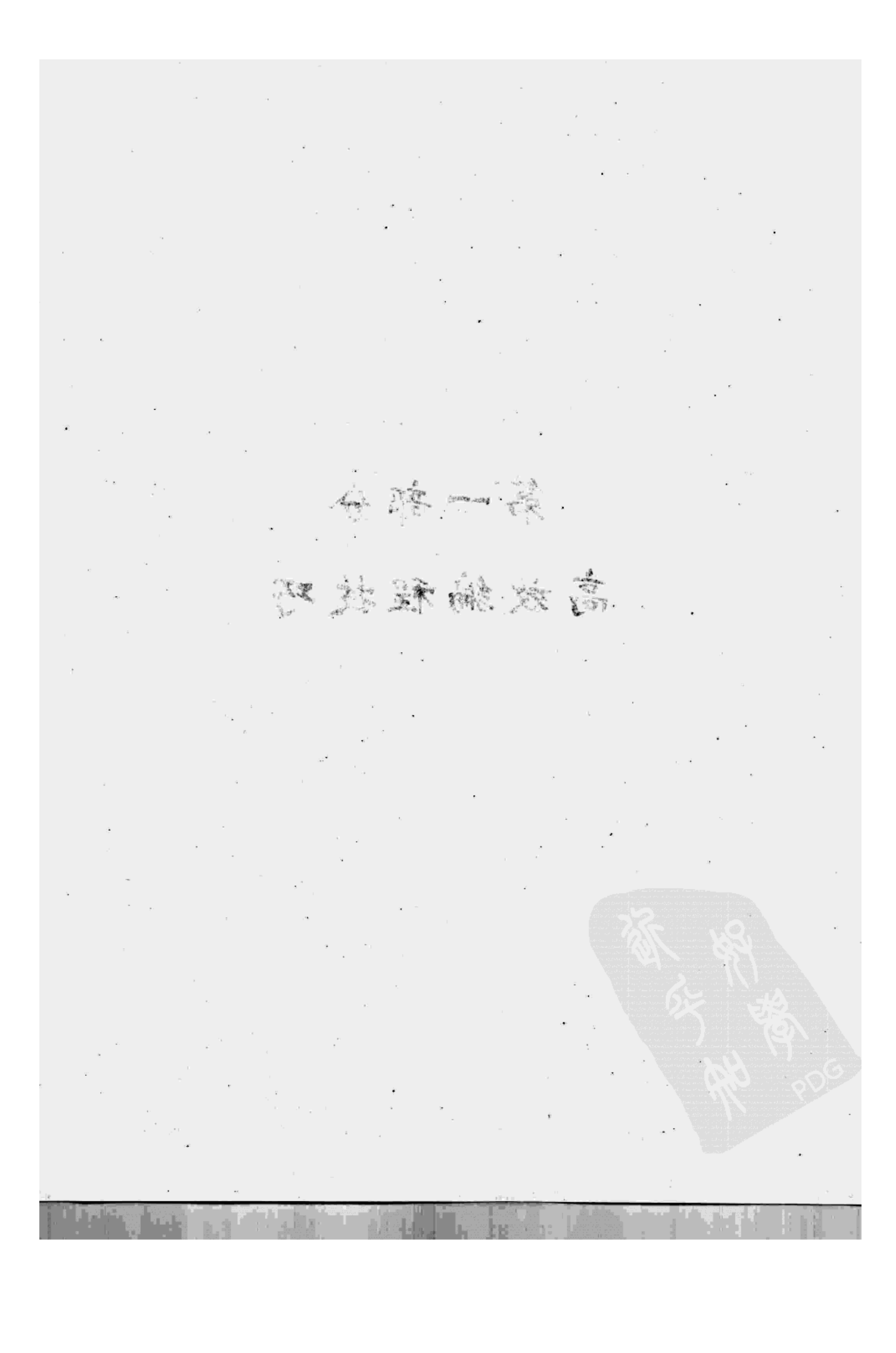
第二部分 案例介绍

案例目录

第一部分

高效编程技巧





第一部分

高效编程技巧

第1章 MATLAB快速入门

本章旨在帮助从未接触过MATLAB的读者能快速入门，对MATLAB产生兴趣，能够用MATLAB完成简单任务，并告之其后续学习方法，确保其能有效的持续提高。

本着让初学者迅速上手的思想，本章由MATLAB界面入手，介绍基本的MATLAB操作环境，进而用一系列短小的程序帮助读者建立起对MATLAB的最初印象，激发学习兴趣。在这之后对M语言进行简明扼要介绍以帮助读者能自己编写MATLAB程序。最后一节是笔者长期使用、学习MATLAB并在国内众多MATLAB论坛与各层次MATLAB使用者交流的经验总结，相信无论读者以前用没用过MATLAB，都能从中得到收获。

1.1 熟悉MATLAB环境

1.1.1 MATLAB的启动

Windows环境下，MATLAB常用启动方法有以下几种：

1) 双击桌面上的MATLAB快捷方式图标(一般指向MATLAB安装目录下bin文件matlab.exe文件)；

2) 依次单击Windows任务栏上"开始"按钮→"所有程序"→MATLAB，找到MATLAB R2009a选项(其他版本找到对应选项)进入MATLAB界面；

3) 单击Windows任务栏上"开始"按钮，在弹出的开始菜单中单击"运行"按钮，输入MATLAB并按Enter键进入MATLAB界面。

1.1.2 MATLAB desktop

第一次启动MATLAB完毕后，呈现给读者的是如图1.1所示的MATLAB界面(也即默认状态下的桌面)。对于初学者来说，最主要需要了解以下三个窗口，即命令窗口(Command Window)、工作空间(Workspace)和历史命令(Command History)窗口。

命令窗口是用来输入各种MATLAB命令，观察运算结果的窗口。

工作空间可以用来浏览程序运行过程中或完毕后得到的变量的值，当工作空间有变量后，可以选中要观察的变量，双击它，MATLAB会新开一个窗口来展示这个变量的值或者属性。

历史命令窗口里面记录了曾经在Command Window中输入过的命令。当选中的是历史命令窗口或者命令窗口时，可以按键盘上的上、下方向键来选中或者调出上一条、下一条指令。双击历史命令窗口中某条指令，则会在命令窗口中重新运行这条指令。

由于本书的侧重点不在基础知识上，限于篇幅，关于这些窗口的详细介绍就略去了。需要说明的是，MATLAB这些窗口操作非常人性化，人机交互性非常好。读者如果初次使用MATLAB，可以按照自己的Windows系统使用经验来试探着操作这些窗口、按钮，观察结果，

很快就会熟悉MATLAB基本的操作环境。如果对某些图标的意义不清楚，可以把鼠标指针分别移到相应的图标上面（不要点击），过一两秒钟，就会显示当前图标的功能介绍。如果在操作过程中关闭了某些窗口，譬如命令窗口、工作空间等而改变了桌面结构，想要恢复默认桌面显示状态，只需单击菜单栏里Desktop菜单项，进入后单击Desktop Layout，选择Default就恢复默认桌面结构了。

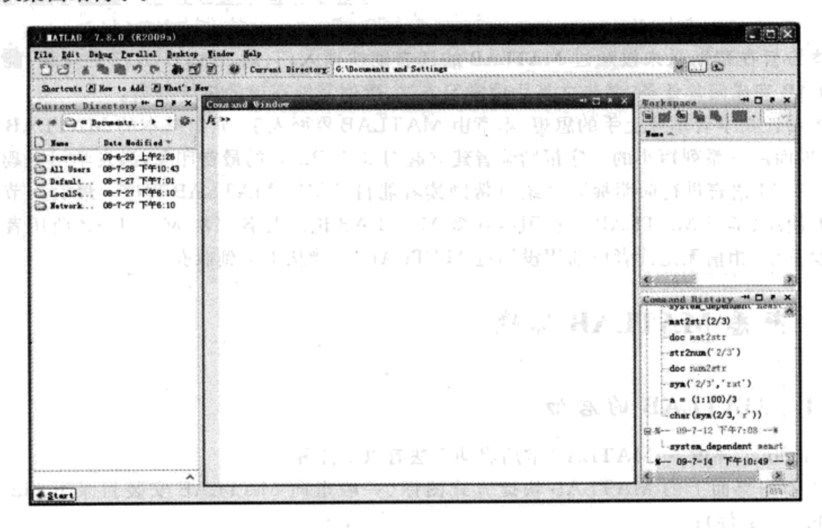


图1.1 MATLAB R2009a默认桌面

1.1.3 MATLAB程序编辑器(Editor)

MATLAB中M文件的编写都在程序编辑器里进行。一个完整的MATLAB项目或者工程可能由一个或者多个M文件构成，而一个M文件中可能包含一个或者多个各种类型的函数。

打开Editor的常用方法有如下几种：

1) 双击磁盘上现有的M文件；

2) 命令窗口中输入edit并按Enter键；

3) 单击MATLAB主窗口上方工具栏中□图标；

4) 依次选择MATLAB主窗口上方菜单栏：File→New→Function M-File。

1.2 MATLAB牛刀小试

1.2.1 Hello,MATLAB

现在让我们来看MATLAB是如何实现"Hello,World!"这样简单的程序的。MATLAB实现起来非常方便，一般常用的有两种方法：



方法 1 在 Command Window 窗口中输入下列代码并按 Enter 键：

```
disp('Hello,MATLAB! ')
```

方法 2 在 Command Window 窗口中输入下列代码并按 Enter 键：

```
A = sprintf('Hello,MATLAB! ')
```

以上两种方法的区别是：第一种方法是直接在屏幕上显示“Hello,MATLAB!”字样；第二种方法是将“Hello,MATLAB!”作为一个字符串常量赋值给变量 A，然后在屏幕上显示 A 的内容，即“Hello,MATLAB!”

1.2.2 万能计算器用法

相信没有用过 MATLAB 的朋友看了这一小节后，肯定会对 MATLAB“电子演算纸”的称谓有较为形象的认识，因为对于简单的计算问题，在 MATLAB 里只是输入计算的表达式然后按下 Enter 键这么简单。

【例 1.2-1】 计算 $\sin(\ln(\pi))^{\exp(1+\sqrt{2\pi/3})}$。

要计算上述表达式的值，只需要在 Command Window 窗口中输入下列代码并按 Enter 键，即得到结果。

```
sin(log(pi))^exp(1 + sqrt(2 * pi/3))
```

命令窗口中运行结果如下：

```
ans =
    0.3388
>>
```

log 是 MATLAB 自带的求自然对数的函数，表达式中的四则运算、指数运算、开方运算等书写形式和大多数其他的编程语言都一致。

ans 是 MATLAB 中的表达式没有指定赋值给其他变量时，用来存放表达式结果的默认的变量，它是 answer 的缩写。

默认状态下，MATLAB 的显示精度是 short 型，如果想查看更多位的有效数字，可以通过运行 format long 来实现。本例中，上述操作的过程和结果如下：

```
>> format long
>> ans
ans =
   0.338830950391107
```

如果想恢复到原来的显示精度，只需要运行 format short 即可。

注意 不要混淆显示精度和计算精度。MATLAB 默认的计算精度是 double 型，而显示精度是呈现结果时候的精度，与计算精度无关。很多初学者认为显示精度是 short 型时，计算精度也是 short 型，这是不对的。

【例 1.2-2】 计算福彩双色球一等奖的中奖概率。福彩双色球玩法如下：从编号 1～33 的红球里任选 6 个，另外在编号 1～16 的蓝球里再任选 1 个。如果选择的红球和蓝球和当期

的开奖结果完全一致(顺序可以不同)则中一等奖。

这是个组合问题，其中奖概率用数学表达式表示为$\frac{1}{C_{33}^{6}C_{16}^{1}}$，用MATLAB解，代码如下：

```
>> p = 1/nchoosek(33,6)/nchoosek(16,1)
p =
  5.6430e - 008
```

分析结果可知，中奖概率只有亿分之五左右。由此可以看出，一等奖对绝大多数人来说是可望而不可即的。

1.2.3 设计一个"囧"的动画

【例1.2-3】 随机展示一下方向、大小、颜色各异的"囧"字。

运行如下代码，观察呈现的动画。

```
function RandDisplayJiong
axis off;   % 隐去坐标轴
set(gcf,'menubar','none','toolbar','none'); % 不显示当前 figure 菜单栏和工具栏
for k = 1:100 % 循环 100 次
% 每次在(rand,rand)这个随机的位置，选择 20～50 之间随机分布的一个数作为其字体大小，以隶书的
% 形式，随机生成 RGB 颜色，并随机旋转一定的角度来显示"囧"字
h = text(rand,rand,...
['\fontsize{',num2str(unifrnd(20,50)),'}\fontname{隶书}囧'],...
'color',rand(1,3),'Rotation',360 * rand);
pause(0.2); % 每显示完一次暂停 0.2s
end
```

图1.2是代码运行完毕最后得到的图片(由于随机，每次运行结果图都不完全一样)。

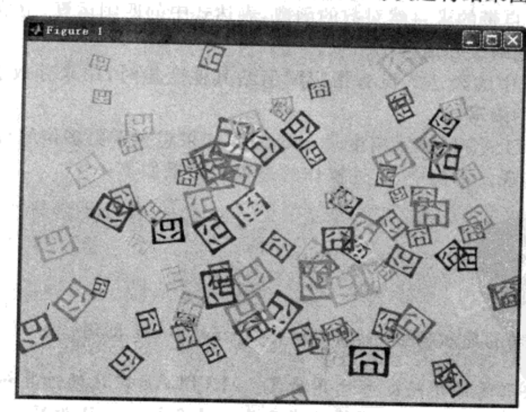


图1.2 "囧"的动画效果图

1.2.4 用MATLAB编写的第一个函数

在本部分将编写一个结构相对完整的MATLAB函数，这个函数包含输入和输出。我们



用这个函数来解决下面这个有趣的问题。

【例 1.2-4】 一只失明的小猫不幸掉进山洞里，山洞有三个门，一个门进去后走 2 h 可以回到地面，从第二个门进去后走 4 h 又回到原始出发点，不幸的是从第三个门进去后走 6 h 还是回到原始出发点。小猫由于眼睛失明，每次都是随机地选择其中一个门走。那么可怜的小猫走出山洞的期望时间是多少？

这个问题如果按常规思路，则需要求几个级数的和，很麻烦。不过可以这么想，设小猫走出山洞的期望时间为 t，如果小猫不幸进了第二个或第三个门，那么它过 4 h 或 6 h 后又和进门之前面临的状况一样了，只不过这两种不幸的情况发生的概率都为 1/3。而万幸一次性走出去的概率也是 1/3。于是可以得到下面的方程：

$$t=2\times(1/3)+(4+t)\times(1/3)+(6+t)\times 1/3$$

解得 $t=12$。

为了验证结果的正确性，下面用 MATLAB 编写模拟小猫 n 次出洞时间的函数，该函数返回的结果 $\boldsymbol{T}$ 为 n 次出洞时间组成的数组，代码如下：

```
function T = cat_in_holl(n)
T = zeros(1,n);
for k = 1:n
    c = unidrnd(3,1);
    while c~= 1
        if c == 2
            T(k) = T(k) + 4;
        else
            T(k) = T(k) + 6;
        end
        c = unidrnd(3,1);
    end
    T(k) = T(k) + 2;
end
```

下面来分析一下这个函数。“function”是函数声明符，其后定义了函数的输出变量，函数名以及输入变量，本例中，“T”为函数输出变量，“cat_in_holl”为函数名，“n”为输入变量。第一行是函数的声明，函数体必须另起一行开始写。T=zeros(1,n)；表示生成一个 1 行 n 列的全 0 行向量，这步主要就是预分配内存，因为最后得到的 T 是小猫 n 次出洞的时间组成的数组，数组长度事先已知。给 T 预分配内存是基本的 MATLAB 高效编程准则之一，关于这一点将在后续章节中详细讨论。

接下来是一个 for 循环体(k 从 1 循环到 n)，for 循环体内部还有一个 while 循环体(当 c ≠1 时，运行 while 循环体内的代码)，while 循环体内还有一个 if-else 选择分支结构。这些循环和选择分支的意义与其他语言都类似，不同的就是格式，每一个循环体或者选择分支结构结束时，MATLAB 用“end”来标志。

unidrnd 是 MATLAB 自带的函数，unidrnd(3,1)表示在 1,2,3 中等概率地随机生成一个数字。关于 unidrnd 函数的详细介绍，读者可以运行 doc unidrnd 查看详细说明。

打开 Editor，将上述代码粘贴过去，保存为 cat_in_holl. m 文件，并使之在 MATLAB 搜索路径上，就可以在命令窗口输入“Z=cat_in_holl(10000)；”来观察模拟小猫 10 000 次出洞的时

间了。可以运行 mean(T)观察平均时间，会发现随着 n 的增长，mean(T)越来越接近 12。

1.2.5 用 MATLAB 运行 Windows 系统命令

在 MATLAB 环境里可以运行 Windows 系统命令，常用的方法有两种：一种调用格式是“! 系统命令”；另一种调用格式是“dos 系统命令”。下列实验基于 MATLAB 2009a 和 Windows XP sp2 操作系统。

读者可以在 MATLAB 命令窗口中输入“!calc”、“!mspaint”、“dos calc”代码，运行并观察结果。因为 MATLAB 可以运行系统命令，我们可以充分利用这个功能。譬如，一个程序要运行很长时间，而我们又不能一直守在计算机前，这时可以在程序运行完并保存好所需要的结果后在程序最后一行加上“!shutdown - s”，则程序运行完毕后会自动关闭计算机。

1.2.6 用 MATLAB 发送电子邮件

下面以一个有意思并且实用的问题来结束本节——用 MATLAB 发送电子邮件。

通过 MATLAB 发送电子邮件的重要意义在于：譬如正在运行一个程序，而这个程序运行时间很长，几小时、几天甚至更长，这期间我们不能一直守在计算机旁，比如要回家，但是又急于用这个程序的结果进行下一步的分析工作，这时候我们希望 MATLAB 能够在程序计算完成时，将需要的东西保存下来，然后以“附件”的形式发送到指定的邮箱，完成后，MATLAB 再运行系统命令关闭计算机。

MATLAB 自带了一个 sendmail 函数，可以用来发送电子邮件。如果用 sendmail 发邮件，只能发到不用 authentication 的地址。可是为了防止垃圾邮件蔓延，互联网上的公共邮箱几乎都要 authentication。那么怎么实现呢？那就是用 gmail 的邮箱，利用 Java 的方式发送到其他邮箱。实现过程举例如下(实验环境 MATLAB R2009a＋Windows XP sp2＋Gmail 的电子邮箱)：

```
function MySendMail
a = rand(100);
DataPath = [matlabroot,filesep,'mydata.mat'];
save(DataPath,'a');
MailAddress = 'Gmail 的邮箱地址';
password = '邮箱密码';
setpref('Internet','E_mail',MailAddress);
setpref('Internet','SMTP_Server','smtp.gmail.com');
setpref('Internet','SMTP_Username',MailAddress);
setpref('Internet','SMTP_Password',password);
props = java.lang.System.getProperties;
props.setProperty('mail.smtp.auth','true');
props.setProperty('mail.smtp.socketFactory.class',...
'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port','465');
subject = 'MATLAB 发的测试邮件';
content = '你好，这份邮件是我用 MATLAB 发的，数据见附件';
sendmail('收件人 Email 地址',subject,content,DataPath);
```



1.3 M语言介绍

1.3.1 数值和变量

1. 数 值

MATLAB里的数值采用十进制表示，并可以用科学记数法表示，对于绝对值小于1的小数，"0"可以省略。以下记数都合法：

5, 0.5, .5, −.5, 0.001, 6.02e23, 2.997 9e8, 6.626e−34, 6.67e−11

MATLAB的数值计算默认是定义在复数域上的浮点计算，采用的标准是IEEE754:1985《二进制浮点运算》，一个非零的浮点数可以表示为

$$x=\pm(1+f)2^e$$

其中：f是尾数($0\leqslant f<1$)；e是指数，默认情况下，MATLAB用double(双精度)型来表示一个浮点数，即64位来表示x，其中f用52位表示，e用11位表示，$-1\,022\leqslant e\leqslant 1\,023$，剩下1位表示$x$的符号。由此可以看出，$f$决定了数值的精度，这个精度是$2^{-52}$，即MATLAB中预定义变量eps的值。$e$决定了数值的范围，其之所以没有取1024和−1023，是因为MATLAB已经为两个特殊类型的数预留了。f取0，e取1024表示的是预定义变量inf，非规范化数字(Denormal numbers)由e取−1023时来表示。

了解了上述定义之后，就可以理解MATLAB中一些预定义变量的含义了，举例如下：

```
eps:2^-52
realmin:2^(-1022)=2.2251e-308
realmax:(2-eps)*2^1023
```

读者如果想更深入地了解MATLAB浮点计算的有关机理，可以参考MATLAB软件创始人Moler教授的《Numerical Computing with MATLAB》一书的第一章。

2. 变 量

很多MATLAB书籍将MATLAB变量分为MATLAB默认的预定义变量(如：pi,i,j,eps,nan,inf等)和用户自己定义的变量。

其实这些所谓的预定义变量是MATLAB内部的一些函数，这些函数被调用后返回其函数名所表示的值，方便起见，人们仍旧称其为预定义变量。这些函数有的没有输入参数，如pi,i,j等，有的可以有也可以没有输入参数，如realmin,realmax,eps,inf,nan等。举例来说，我们直接在命令窗口输入eps并按Enter键，返回的是用double型来表示一个浮点数情况下的精度值，即2^{-52}。如果输入eps('single')按Enter键，那么返回的就是用单精度型表示浮点数情况下的精度值，即2^{-23}。表1-1列出了常用的MATLAB预定义变量，大家注意在编写程序时不要使用这些变量名来表示自己定义的变量。

用户自定义的变量。MATLAB中变量或者函数的命名遵循以下三条原则：

1) 变量名、函数名对字母大小写是敏感的。如变量data和Data表示两个不同的变量，cos是MATLAB定义的余弦函数名，但Cos,COS都不是。

2) 变量名的第一个字符必须是英文字母，其余可以是英文字母、数字或者下画线，最多可包含63个字符(运行namelengthmax命令可以得到这个数)。如a1,a_b_1是合法的变量名，

_a,1a,a$则都不是合法的变量名。

3）变量名中不得包含空格、标点、运算符。

表1-1 MATLAB中常用的预定义变量

预定义变量	含 义	预定义变量	含 义
ans	计算结果的默认变量名	pi	圆周率
eps	特定精度表示浮点数时的精度值	NaN 或 nan	非数(Not a Number)，如0/0，∞/∞
Inf 或 inf	特定精度表示浮点数时候的无穷大	realmax	特定精度表示浮点数时的最大正实数
i 或 j	虚数单位	realmin	特定精度表示浮点数时的最小正实数

1.3.2 MATLAB程序流程控制

和其他大多数编程语言一样，MATLAB主要有以下几种程序流程控制方法，调用格式分别为：

1. if-else-end 结构

第一种：

```
if expr % expr 为表达式，如果成立，则执行到“end”为止的所有 commands，否则不执行 commands
commands;
end
```

第二种：

```
if expr1                 % expr1 成立则执行 commands1；否则判断 expr2；
commands1;
elseif expr2             % 如果 expr2 为真，则执行 commands
commands2;
else                     % expr1, expr2 都不成立，执行 commands3
commands3;
end
```

其中根据程序分支的多少，“else-if”的个数可以有0个到多个。

2. switch-case-otherwise-end 结构

```
switch expr                  % expr 为表达式
  case value1                % expr 取值为 value1 时，执行 commands1
    (commands1)
  case value2                % expr 取值为 value2 时，执行 commands2
(commands2)
  case valuek
    (commandsk)
  otherwise                  % expr 取值为 value1 到 valuek 之外的任何值时，执行 commands
    (commands)
end
```

“otherwise”类似于“if-else-end”结构中的“else”，视程序逻辑需要，并不是一定要有，

但为了程序结构上的完整性以及某些情况下调试程序的需要，建议始终保留。commands 也可以为空语句。

3. for 循环

```
for ix = array
commands;
end
```

在命令窗口中运行下列代码，体会 for 循环的用法。

```
for ix = 1:10
a = ix;
end

for ix = (1:10)'
a = ix;
end

for ix = [1 2 3;4 5 6;7 8 9]
a = ix;
end
```

运行上面代码可以发现，MATLAB 的 for 循环的机制是遍历 array 的列，无论这个 array 是向量还是矩阵。如果 array 是行向量，那么 for 就遍历它的每个元素；如果是列向量，for 循环就循环一次，即遍历列向量自身；如果 array 是矩阵，那么 for 循环就遍历它的每一列，循环 n 次，n 是 array 的列数。此外，如果 array 是三维矩阵，那么 for 循环先遍历第一页的所有列，之后是第二页的所有列，等等。

4. while 循环

```
while expr
commands;
end
```

while 循环的机制是当 expr 为真的时候执行 commands 命令，直到 expr 为假。所以执行 commands 命令必须得在有限循环次数内使得 expr 为假，否则 while 循环会一直循环运行下去。

5. try－catch 结构

```
try
  commands1          % Try block
catch ME
  commands2          % Catch block
end
```

该结构意义是执行 commands1，如果不发生错误，则不用执行 commands2；如果执行 commands1 的过程中发生错误，那么 commands2 就会被执行，同时，ME 记录了发生错误的相关信息。

6. 其他中断、暂停语句

MATLAB在循环体内还可以利用continue语句跳过位于它之后的循环体中的其他指令，而执行循环的下一个迭代。例如：

```
for ix = 1:5
    if ix == 3
        continue;
    end
    disp([' ix = ',num2str(ix)])
end
```

运行结果：

```
ix = 1
ix = 2
ix = 4
ix = 5
```

MATLAB可以用break语句结束包含该指令的while或for循环体，还可以在if - end，switch - case，try - catch结构中导致程序中断。例如：

```
ix = 3;
jx = 6;
    if ix == 3
        disp([' ix = ',num2str(ix)]);
        break
        disp([' jx = ',num2str(jx)]);
    end
```

运行结果：

```
ix = 3
```

可见break中断程序执行了，导致“jx=6”没有被显示。

pause(n)可以使程序暂停n秒后再继续执行；pause指令使程序暂停执行，等待用户按任意键继续；return指令可以结束“return”所在的函数的执行，如果“return”所在函数由函数fun来调用的，则程序将控制转至函数fun，如果“return”所在函数是在命令窗口直接运行的，则程序将控制转至命令窗口。

1.4 学习MATLAB的方法

通过前面几节的介绍，相信刚接触MATLAB的读者对MATLAB已经有了初步的了解。MATLAB功能非常强大，前面介绍的仅仅是其全部功能的非常小的一部分。MATLAB本身自带有几十个涉及多个专业领域的工具箱，每个工具箱都包含很多解决专业问题的函数，这些函数都是建立在一篇篇经典的论文基础上，而实际学习中没必要也不可能对每个工具箱中的每个函数的用法都掌握得很精通。那么我们怎么来学习MATLAB呢？笔者给出的建议是：



用好 help,学会搜索,多读高手的程序,自己多练习。

第一条:用好 help。古语说:"授之以鱼不如授之以渔"。MATLAB 的帮助系统做得非常好,非常完善,帮助文档对 MATLAB 每个函数的用法都进行了详细的描述。学会使用 help 后,就好比学会了"打鱼"的本领,不仅可以大大减轻记忆的负担,而且可以帮助自身持续提高 MATLAB 应用水平。

这里建议大家牢记几个关键词,有了它们,相信很多时候就掌握了解决问题的方法。这些关键词即:help 或者 doc, see also, help navigator, Tab。不要小瞧这几个词,可以毫不夸张地说,这几个词带给读者的帮助是随着读者的 MATLAB 应用水平不断提高而越来越大的。

运行" help 函数名",会在 command window 窗口里列出该函数的详细介绍,包括调用格式,输入输出参数的意义等。运行"doc 函数名",会调出该函数在帮助文档里介绍的那一页,可读性比"help 函数名"要好些,具体使用哪个就看个人使用习惯了。

在当前查询的函数的最后,MATLAB 会在"see also"后面列出一些功能上和被查询的函数有相同或者相似地方的一些函数。不要小瞧这个"see also",很有可能被查询的函数完成不了预期的任务,而实现相关功能的函数就在 see also 里。

help navigator,堪称 MATLAB 中的"百度",打开帮助文档后其位于界面的左半部分。往往人们在对一个问题毫无头绪的时候,可以用它来搜一下相关的关键词,很多时候会有意想不到的结果。譬如,如果想知道 MATLAB 中有无求集合交集的函数,可以在 help navigator 里的搜索栏中输入集合交集的两个关键词"set intersect"(注:如果在关键词之间加"-"连接号,表示搜索时将这些关键词作为一个短语整体来搜,而不是各自独立搜),按 Enter 键后发现,搜索结果第一项——intersect 函数的介绍:"Find set intersection of two vectors"即求集合交集的函数,而在 see also 里我们还一并得知了求集合并集、补集等函数,收获颇丰。

Tab 键也非常有用。当我们对一个函数记不清楚的时候(譬如只记得前半部分某几个字母),我们写下这几个字母并按 Tab 键,MATLAB 会列出所有刚写的那些字母开头的函数。我们找到需要的函数后可以进一步在帮助文档中查看其用法。

总之,建议读者经常翻看帮助文档,积累对 MATLAB 学习很重要,经常翻看帮助文档,会不知不觉地积累很多 MATLAB 相关技巧、知识。

第二条:学会搜索。当今社会的信息量极其丰富,如何获取我们需要的信息,搜索能力的重要性不言而喻。

对于 MATLAB 学习来说,搜索能力也是非常重要的。其实在第一条中已经对搜索能力有要求了,即在 help navigator 中搜索我们需要的函数。对于 MATLAB 初学者来说,面临的问题通常都早已被解决过很多次了,互联网上有很多现成的答案。可是在各个 MATLAB 社区中,笔者经常看到一些解决过好多次的问题一次又一次地被问起,这从一个侧面反映了有相当一部分 MATLAB 学习者缺乏搜索能力。将问题的解答寄希望于别人总没有通过自己搜索并努力钻研解决来得印象深刻。

搜索的关键在于搜索的关键词。一般说来,关键词不应过长,要有代表性,在选择搜索的关键词时,多想想别人问与自己同样的问题的时候可能会用到哪些关键词,并可以在网上看一些有关搜索技巧的文章。养成这样的习惯,久而久之,搜索经验越来越丰富,搜索能力就会有所提高。

第三条:多读高手的程序。俗话说得好,和臭棋篓子下棋,越下越臭。学习编程也是如此。

MATLAB是一门灵活性非常高的语言，针对同一个问题，不同的人写出来的程序运行效率可能差别巨大。有些人不注意基本的MATLAB编程规范，写出来的代码运行效率低，还抱怨MATLAB运行太慢；有些人把MATLAB运用的得心应手，充分利用MATLAB向量化编程优点，对很多问题能写出不输于C/C++语言，甚至比C/C++语言还快的程序。

读高手的程序，能避免走很多弯路，而且还能享受MATLAB向量化编程带来的便利、高效以及乐趣。找到高手很简单，很多高手都在不止一个MATLAB技术论坛注册了同样的ID，去各大MATLAB技术论坛翻看一下帖子，很容易就能发现哪个ID水平比较高，找到之后可以利用各论坛的搜索功能搜索一下他的帖子，搜出来帖子后就慢慢去读，从中可以学到很多东西。对于初学者，可能会发现很多问题都看不懂，这很正常，谁都有这个过程，重要的是现在能看懂的那部分，有收获的那部分。看完这部分可以再看些通过断点调试，查帮助文档能看明白的。实在看不懂的就先放下，过段时间自己感觉有进步了再回头看看，慢慢得会发现，每看一遍，自己就会有一些新的体会，当通过自己的努力看了很多遍的时候，也许突然就会有豁然开朗、峰回路转的感觉，这种感觉非常棒，会给自己增添不少学习信心以及乐趣。

建议初学者开始不要读比较长的程序，可以挑精简的程序去慢慢研究领会，这样容易保持旺盛的学习兴趣。

最后，说一下亲自动手练习的重要性。对于任何语言来说，写程序都是不断地自我完善的过程，上来就写出没有bug的代码，对于稍微复杂的问题来说都是不现实的。所以不要怕错误，一定要自己多动手编写程序，不要以为看懂书上的代码了，自己就能编出来，只有自己亲自写代码并运行成功，才能切身体会到很多只是看程序所体会不到的东西。如果觉得自己看明白了，可以试着不看书，自己重新写一下，然后和源程序对照一下。

在MATLAB相关技术论坛试着回答他人提出的问题，也是一种被证明了的比较有效的学习方法。在回答他人的问题过程中，自己会去查Help帮助文档，搜索，思考，然后自己再动手，还能参考别人给出的答案。这样一来，就把上面提到的几点学习方法有机结合在一起，久而久之，慢慢积累，学得东西越来越多，水平就会逐渐提高。在互助的过程中还能结识一些有着共同兴趣爱好的朋友，大家彼此交流，共同进步。

第2章 重新认识向量化编程

从本章开始，进入本书的主体部分，所讨论的话题很多时候都是围绕着“如何写出高效代码”这个主题，而对一些基本的 MATLAB 知识就不再做过多的陈述。

2.1 向量化编程流行的一些观点

MATLAB 向量(矢量)化编程，是一个伴随 MATLAB 发展的永恒的话题，是 MATLAB 语言的精髓所在，向量化编程运用好了，可以从代码运行效率明显改善中获得成功的快乐。传统的流行观点大致如下：

1) 尽量避免循环的使用，多使用 MATLAB 的内置函数。
2) 能用逻辑索引解决的就不用数值索引。
3) 使用变量前养成预分配内存的习惯。
4) 向量化计算代替逐点计算。
5) 能用普通数值数组完成的工作尽量不用 Cell 型数组。
6) 如果矩阵含有大量 0 元素，尽量采用稀疏矩阵来提高运算速度和减少存储空间。

MATLAB 在不断发展，随着版本的升级，有些向量化编程所描述的观点基本没变，但是有些观点就需要修正了。如果还按原来的观点为指导进行编程，那么很多情况下并不能得到效率明显的改善，甚至还不如按原来观点所不提倡的编程方式编写的代码效率高。

高版本的 MATLAB 向量化编程是一个比较复杂的问题，遵循的原则之间都有互相制约、相辅相成的关系。这些需要读者在实际应用中不断体会、感悟。下节开始，将讨论高版本的 MATLAB 高效编程注意事项。

2.2 重新认识循环

2.2.1 高版本 MATLAB 对循环结构的优化

从 MATLAB 6.5 版开始，MATLAB 引入了 JIT(just in time)技术和加速器(accelerator)，并在后续版本中不断优化。到了 MATLAB R2009a，很多情况下，循环体本身已经不是程序性能提高的瓶颈了，瓶颈更多的来源于循环体内部的代码实现方式，以及使用循环的方式。循环就是多次重复做同一件事，如果这件事本身的代码写得不优化，放在循环体内多次实现后必然造成运行时间过长。

老版本的 MATLAB 对循环机制的支持不好，所以提倡避免循环，而高版本的 MATLAB 对循环机制的支持大大提高了，因此就不必再像过去那样谈“循环”色变了，不用千方百计避免循环了。当使用循环造成程序运行时间过长时，不要武断地将代码运行效率低归结到使用了循环。有的时候，我们千方百计地把一段代码向量化了，凭自己编程经验(很多是使用老版本积累下来

的经验）和常识（从老的教科书得到的常识）觉得程序很标准、很优化。殊不知，实际测量时会发现其性能不比采用很自然的想法实现的程序效率高多少，甚至还会降低。看下面这个例子：

【例 2.2-1】 运行下面的测试 JIT/ accelerator 的代码，体会高版本的 MATLAB 对循环的加速。

```
function JITAcceleratorTest
u = rand(1e6,1); % 随机生成一个 1 * 1000000 的向量
v = zeros(1e6,1);
tic
  u1 = u + 1;
time = toc;
disp(['用向量化方法的时间是:',num2str(time),'秒！']);
tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['循环的时间是:',num2str(time),'秒！']);

feature jit off;
tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['只关闭 jit 的时间是:',num2str(time),'秒！']);

feature accel off;

tic
for ii = 1:1000000
    v(ii) = u(ii) + 1;
end
time = toc;
disp(['关闭 accel 和 JIT 的时间是:',num2str(time),'秒！']);

feature accel on; % 测试完毕重新打开 accelerator 和 JIT
feature jit on;
end
```

运行结果如下：

```
用矢量化方法的时间是:0.0095308 秒！
循环的时间是:0.010176 秒！
只关闭 JIT 的时间是:0.084027 秒！
关闭 accel 和 JIT 的时间是:1.2673 秒！
```

自从引入 JIT 和 accelerator 后，MATLAB 对这两项功能默认都是打开的，这也是高版本 MATLAB 对循环支持好的原因。关闭 JIT 和 accelerator 需要用到 MATLAB 一个未公开



(undocumented)的函数：feature。feature accel on/off 即为打开/关闭 accelerator，类似的打开/关闭 JIT 是 feature jit on/off。

上面代码是计算一个长向量与一个标量的和，我们会发现，当 JIT 和 accelerator 都打开的状态下，循环和矢量化运算所需要的时间从统计意义上来讲，已经没有显著差别了。关闭 JIT 后，运行时间变为原来的 8 倍左右，而再关闭 accelerator，运行时间立刻变为原来的 100 多倍。

再来看一个例子：

【例 2.2-2】 由一个 m×n 的矩阵构造一个 m×(m+n−1)的矩阵。构造方式如下：以 4×4矩阵 ***A*** 为例，构造目的矩阵 ***B***：

```
A =
        1     2     3     4
        3     4     5     6
        2     3     4     5
        1     3     4     6
B =
    1     2     3     4     0     0     0
    0     3     4     5     6     0     0
    0     0     2     3     4     5     0
    0     0     0     1     3     4     6
```

例 2.2-2 中所涉及的问题用循环来解决非常容易实现。我们这里想要讨论的是，用向量化的方法解决该问题与用循环解决该问题两种运行时间的对比(还不考虑写出向量化的代码比用循环实现多花的时间)。

向量化思路：观察矩阵 ***B***，发现，如果按行数的话，矩阵 ***B*** 的元素排列顺序是原来矩阵 ***A*** 中的每一行和下一行之间以四个 0 相隔。这样可以计算出矩阵 ***A*** 中的元素在矩阵 ***B*** 中相应的索引值(当然是按行)I，那么可以生成一个和矩阵 ***A*** 大小一样的全 0 矩阵 ***B***，然后令 ***B***(I)＝a(:)；最后注意到 MATLAB 是按列的顺序遍历元素的，所以最后再将矩阵 ***B*** 转置。写成代码就是：

```
function B = rowmove(A)
  [m,n] = size(A);
I = repmat(1:n,m,1) + repmat((0:m-1)'*(m+n),1,n);
B = zeros(m+n-1,m);
B(I(:)) = A(:);
B = B';
```

循环的代码如下：

```
function C = LoopRowMove(A)
  [m,n] = size(A);
C = zeros(m,m+n-1);
for k = 1:m
C(k,k:k+n-1) = A(k,:);
end
```

可以看出，循环的思路以及操作非常简单，就是循环赋值操作。下面随机生成一个 1000×1000的矩阵，来用上述两种方法来比较一下运行速度：

```
A = rand(1000);
tic;B = rowmove(A);time = toc;
disp(['向量化求解时间是:',num2str(time),'秒！'])
tic;C = LoopRowMove(A);time = toc;
disp(['用循环求解时间是:',num2str(time),'秒！'])
if isequal(B,C)
    disp('两种方法结果完全一样')
end
```

运行结果如下：

```
向量化求解时间是:0.11389 秒!
用循环求解时间是:0.054231 秒!
两种方法结果完全一样
```

循环的时间反而不到向量化时间的一半，而且循环的代码要比向量化的代码容易写得多，算上“开发”时间，本例用循环比用向量化要高效的多。这是为什么呢？因为本例循环结构内部的代码仅仅为“赋值”这样简单的操作，不存在函数调用。目前在高版本的 MATLAB 中，循环本身往往不是程序瓶颈，反而函数调用产生的额外开销在很多情况下是构成程序瓶颈的因素之一，尤其是采用大量低效率的函数结构时(譬如 inline 对象，这个将在后续章节详细讨论)。

反观向量化操作，调用了两次 repmat 函数，至于 repmat 内部又做了些什么？有兴趣的话，可以运行 edit repmat 命令，查看 repmat 的源代码，会发现，里面有很多行代码都是这个问题没必要执行的。事实上，为了保证每个函数具有尽可能广的适应面，Mathworks 开发人员在编写函数时一般都在函数入口处做很多层判断，确保不同的输入能正确地执行相应的语句。

图 2.1 所示的窗口是用 MATLAB 的 Profiler 工具剖析“B=rowmove(A);”这条语句的结果。可以看出，两次调用 repmat 函数所占时间比例最高，其次是转置操作，这两项一共占去了整个程序的运行时间的 62%，而这两项对于本例来说完全都可以避免。而用循环实现时，只需要赋值 m 次即可，所用的系统开销相对要少。

Profiler

File Edit Debug Desktop Window Help

Start Profiling Run this code: B=rowmove(A); Profile time: 0 sec

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
3	I=repmat(1:n,m,1)+repmat((0:m-...	1	0.045 s	36.8%	
6	B=B';	1	0.032 s	25.6%	
5	B(I(:))=A(:);	1	0.031 s	24.8%	
4	B=zeros(m+n-1,m);	1	0.016 s	12.8%	
2	[m,n]=size(A);	1	0 s	0%	
All other lines			0.000 s	0.0%	
Totals			0.123 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot

图 2.1　剖析例 2.2-2 所用的向量化方法性能

一些读者在使用 MATLAB 的过程中对向量化的高效率体会颇深，加上传统的书籍以及网络上流传的传统观点对循环的低效率的描述，导致其逐渐形成一种观点，就是只要是循环，效率就是低的。从上面的例子可以看到不能一味地否认循环，下节会讲到，MATLAB 效率低往往是由于发生大量函数调用以及算法本身运算次数多引起的。譬如计算多项式：

$$p_4(x)=a_1x^4+a_2x^3+a_3x^2+a_4x+a_5$$

如果直接按照上述表达式按部就班进行的话，我们需要计算 4+3+2+1=10 次乘法以及 4 次加法，如果用下述表达式计算：

$$p_4(x)=((((a_1x+a_2)x+a_3)x+a_4)x+a_5$$

则只需要计算 4 次乘法和 4 次加法，随着多项式次数的增加，差距将越来越明显。请看下面的例子。

【例 2.2-3】 计算 $\sum_{k=1}^{10^6+1}k(1+10^{-6})^{10^6+1-k}$。

对于本例可以利用 sum 函数，polyval 函数以及根据前述将多项式拆开的算法自行写代码来实现：

```
clear
N = 1e6 + 1; k = [1:N]; x = 1 + 1e - 6;
tic
p1 = sum(k. * x.^[N - 1: - 1:0]);  % 方法 1:用 sum 函数的方法
p1, toc
tic, p2 = polyval(k,x), toc        % 方法 2:调用 polyvol 函数实现
tic, p3 = k(1);
for i = 2 : N                      % nested multiplication
p3 = p3 * x + k(i);
end
p3,toc                             % 方法 3:根据上述多项式求和算法实现
p1 =
   7.1828e + 011
Elapsed time is 0.167499 seconds.
p2 =
   7.1828e + 011
Elapsed time is 0.039696 seconds.
p3 =
   7.1828e + 011
Elapsed time is 0.014916 seconds.
```

从上述运行结果来看，根据多项式求和算法自己编写并且还用到了循环的代码，所用的时间最短。值得一提的是：polyval 函数核心部分也是采用上述方法 3 中的多项式求和算法，但是为什么时间要稍长一些呢？这是因为 polyval 函数比较通用化，里面有一些额外的代码考虑的是各种可能出现的情况，感兴趣的读者可以在命令窗口中运行“type polyval”查看其源代码运行即可明了。关于这方面的代码运行效率问题 2.3 节还会涉及。

下面再看一个算法本身引起的效率差异的例子，该例也是被很多网友经常问到的问题。

【例 2.2-4】 如何从 1,2,…n 这 n 个数字中随机选出 m 个不重复的数字。

一个比较容易想到的办法就是利用 randperm 函数，随机排列 1,2,…n 这 n 个数字，然后

取前 m 个。可是这样做在 n 较大而 m 较小时效率不高，因为只需要 m 个不重复的数字，却把所有的数字都重新排列了一下。可以利用下面的算法：

令 $\boldsymbol{a}=[1,2,\cdots,n]$，对于 i 从 1 循环到 m，随机等概率地生成一个属于 i 到 n 的随机整数 ind，交换 $a[i]$ 和 $a[ind]$ 的值，循环完毕后取 $a[1]\sim a[m]$。

上述算法写成代码如下：

```
function r = randnchoosek(n,m)
% n:数组,需要从中随机选取 m 个不重复的元素
% r: 数组 n 中随机选取的 m 个不重复的元素
ln = length(n);
for i = 1:m
    ind = i - 1 + unidrnd(ln - i + 1);
    a = n(ind);
    n(ind) = n(i);
    n(i) = a;
end
r = n(1:m);
```

对于 $n=10^6$，$m=1000$ 的情况，我们可以比较两种方法的运行速度：

```
clear;tic
N = 1e6;m = 1000;
rnN = randperm(N);
r1 = rnN(1:m);toc
tic;r2 = randnchoosek(1:N,m);toc
Elapsed time is 0.233197 seconds.
Elapsed time is 0.038107 seconds.
```

可见第二种方法比第一种快得多。

2.2.2 选择循环还是向量化

关于该用循环还是向量化，这是一个比较复杂的问题。笔者结合多年使用 MATLAB 经验给出的建议如下：

(1) 凡是涉及矩阵运算的时候，则尽量用向量化

这是因为，向量化编程是 MATLAB 语言的精髓，如果不熟悉其向量化编程的方法，则相当于没掌握这门语言。MATLAB 以矩阵为核心，MATLAB 卓越的矩阵计算能力是建立在 LAPACK 算法包和 BLAS 线性代数算法包的基础之上的。这两个算法包里的程序都是由世界上多个顶尖专家编写并经过高度优化了的程序。譬如算矩阵乘法的时候，就不要用循环了，因此时循环方法不仅运行效率大大低于 MATLAB 的矩阵乘法，而且开发效率也非常低。

需要说明的是，本原则是建立在参与运算的矩阵不太大，能够适应系统物理内存的条件下。

(2) 如果向量化可能导致超大型矩阵的产生，使用前要慎重

向量化操作获得的高效率很多时候是以空间换时间实现的。

具体说来就是把数据以整体为单位，在内存中准备好，从而使得 MATLAB 内建的高效函数能批量处理。如果处理的数据量很大，譬如一些图像数据、地质数据、交通数据等等，那么可



能导致其准备过程以及存储这些矩阵耗费数百兆乃至千兆(吉)甚至更多的内存空间,如果超过系统物理内存空间,那么会造成效率的低下。原因如下:

第一,数据准备过程也是需要耗费资源的,事先存储大规模的矩阵,势必造成留给 MATLAB 计算引擎乃至操作系统的物理内存大大减少,如果系统内存不够大,势必大大影响计算效率。

第二,如果参与运算的矩阵尺寸过大,可能导致虚拟内存的使用,那么当对这块矩阵进行读写和计算时可能涉及频繁的内存与外存交换区的输入/输出,会造成效率的急剧下降。这时应该对程序进行重新思考和设计,后面章节会就 MATLAB 如何有效处理大规模矩阵进行专门讲解。

(3) 向量化的使用要灵活,多分析其运行机制,并与循环做对比

在网上经常看到一些 MATLAB 使用者视 MATLAB 的循环为"洪水猛兽",千方百计避免循环,不管什么程序都要想尽办法将其向量化。好像只要向量化,速度就会神奇的上去,可是,向量化有些时候会增加实际运算次数,这往往出现在那些不适合向量化的过程中。这样,即使绞尽脑汁、生搬硬套地利用一些向量化技巧、向量化的函数让操作变成矩阵运算,但是增加的无用计算使得即便是更高效的引擎也无法挽回损失。例 2.2-2 就很好地证明了这一点。

(4) MATLAB 初学者要尽量多用向量化思路编程

MATLAB 是一门灵活性与高效性结合的非常好的语言。我们不能过分强调、夸大向量化编程的优势,当然更不能把 MATLAB 当成 C/C++等语言来使用。MATLAB 初学者一定要提醒自己,处处想尽办法矢量化编程,而暂时先不管矢量化后会不会得到性能大幅提高或者应不应该向量化。

这是因为,只有当你充分熟悉 MATLAB 语言特点并且熟悉其最常用的内置函数之后才能灵活运用向量化和非向量化。向量化编程往往需要程序编写者熟悉 MATLAB 的常用内置函数、常用技巧,并且在向量化一个程序的过程中有效锻炼其观察、分析、概括、抽象、归纳的能力。

经过大量向量化编程的训练后,MATLAB 学习者能够更加深入、细腻地感受 MATLAB 语言的优缺点,能够积累大量的内置函数使用经验以及常用的向量化技巧。既能从高效、优美、简洁的向量化代码中获得意想不到提速带来的快乐,也能遇到经过向量化反而没有提高效率的困惑,从而迫使自己进一步加深对 MATLAB 语言的了解。

只有当对 MATLAB 向量化函数、技巧都了解的时候,才能为以后高效的利用 MATLAB 编程打下坚实基础。

(5) 高效的编程包括高效的开发和高效的运行

我们应该明白自己的程序目的是要干什么。如果仅仅是验证某个东西,程序最多运行几次以后就不会再用了。那么这个时候高效的编程就是怎么能够尽快实现程序的功能,让程序尽快在可接受的时间内跑出正确的结果。

如果我们采用自己容易想到的思路实现这个程序,程序运行几秒就可出结果。而我们为了优化这个程序多花的时间远远大于优化后的程序所节约的时间,这是得不偿失的。

2.3 提高代码效率的方法

2.3.1 预分配内存

当矩阵较大时,预分配内存(preallocation),是 MATLAB 高效编程中最应该注意,但其也

是很容易把握的一个技巧。看下面的例子：

【例 2.3-1】 比较预分配和不预分配内存对程序运行速度的影响。

```
function PreAllocMemVSnot
n = 30000;
tic;
for k = 1:n
    a(k) = 1;
end
time = toc;
disp(['未预分配内存下动态赋值长为',num2str(n),'的数组时间是:',num2str(time),'秒！'])
%%
tic
b = zeros(1,n,'double');
for k = 1:n
    b(k) = 1;
end
time = toc;
disp(['预分配内存下赋值长为',num2str(n),'的数组时间是:',num2str(time),'秒！'])
```

运行结果如下：

```
>> PreAllocMemVSnot
未预分配内存下动态赋值长为 30000 的数组时间是:0.58989 秒!
预分配内存下赋值长为 30000 的数组时间是:0.0004281 秒!
```

速度相差 1000 倍之巨！例子中仅仅是一个长为 30 000 的数组，数组的长度越大，这种差距越明显。造成上述速度巨大差距的原因是什么呢？原因是 MATLAB 动态扩充数组在内存中是按图 2.2 所示的过程进行的。

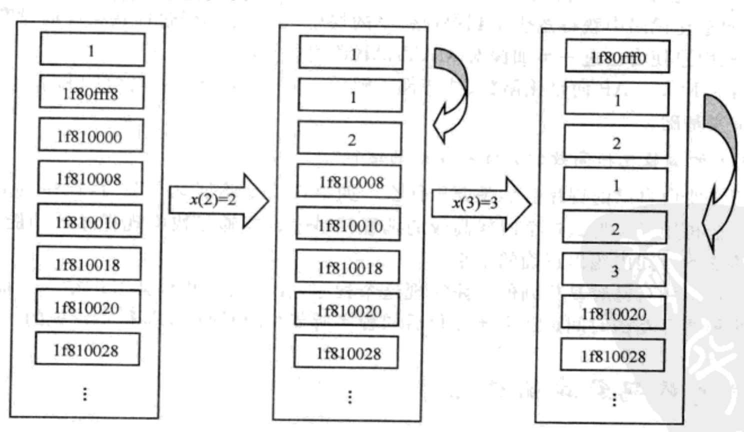


图 2.2 MATLAB 动态扩充数组内存实现方式

图 2.2 描述的是如下代码在内存中的实现方式：

```
>> x = 1;
>> x(2) = 2;
>> x(3) = 3;
```

图中的16进制数字代表内存地址，现在以1,2,3来依次给x动态扩充赋值，由于MATLAB默认是以双精度来表示数字的，因此每个数字占8字节(8B)。可以看出，每次赋值，MATLAB都会在内存中重新开辟一块和当前赋完值之后的x的尺寸相等的连续区域来存储x，并把原来x存储的元素赋值到新的相应区域。可以想象，当动态赋值一个大型数组时，会有大量的时间耗费在无谓的内存读写上，数组越长，耗费的时间增长得越快。而事先向量化呢？即

```
>> x = zeros(3,1,'double');
>> x(1) = 1;
>> x(2) = 2;
>> x(3) = 3;
```

图2.3展示了这一过程。

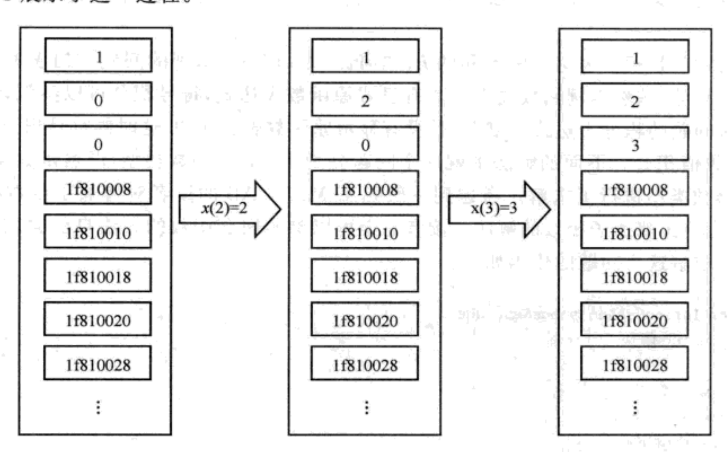


图2.3　MATLAB预分配内存后赋值实现方式

由图2.3可以看出，预分配内存后，不会有时间浪费在无谓的内存读写上，效率得到了大大提高。

细心的读者可能已经注意到了前面预分配内存的程序，譬如b=zeros(1,n,'double')；和我们常见的预分配内存zeros用法有些不同，多了一个数值类型输入参数，其实这是MATLAB R2009a又一个改进的地方。很多人没有注意这点，b=zeros(1,n,'double')或者b(1,n)=0效率要远远高于b=zeros(1,n)，这是因为前者并不在物理内存中真正写入这样一个矩阵，而是在MATLAB进程的虚拟内存空间中声明所需要的一片连续地址，当对生成的矩阵进行操作时，写入物理内存操作才真正发生；而运行后者b=zeros(1,n)，在MATLAB的虚拟内存空间中声明以及写入物理内存都会发生，所以效率比较低。有兴趣的读者可以用R2009a版本试试下面的代码(多运行几次看时间差别)：

```
clear;
tic;a(10000,5000) = 0;toc;clear
tic;a = zeros(10000,5000);toc;clear
tic;a = zeros(10000,5000,'double');toc;
```

经过笔者验证，包括 MATLAB 2008a 在内的以前的 MATLAB 版本都不具有这个性质，笔者没有验证 MATLAB R2008b 版本，有兴趣的读者可以自行去验证。

2.3.2 选用恰当的函数类型

MATLAB有多种函数类型，不同的函数类型调用效率差别相当大，为了写出高效的MATLAB代码，我们要选用恰当的函数类型来完成我们的任务。下面将以一个实用的例子简要介绍一下 MATLAB 常用的一些函数类型，并比较其效率，相关函数类型的详细介绍将放在后面章节进行。

【例 2.3-2】 求 $f(k)=\int_0^5 \sin(kx)x^2\mathrm{d}x$，$k$ 取[0,5]区间不同值时的函数值，并画出函数图像。该例的意义在于其求解方法对于一些复杂的、无显式积分表达式的带参数积分问题具有通用性。

本例关键就是积分表达式中不同的 k，如何让 MATLAB 识别的问题。初学者对这个问题爱用符号积分，诚然本例函数关于 x 具有显式原函数表达式，符号积分可以奏效，但是现实工程中很多问题的积分表达式很复杂，是没有显示原函数表达式的，这时候符号积分就行不通了，只能用数值积分。下面的解法中就这个问题分别用 inline 函数类型、匿名函数类型、嵌套（nested）函数类型进行了求解。考虑到一般熟悉 MATLAB 的读者对普通子函数都比较了解，因此没有给出普通子函数的解法。读者完全可以参照嵌套函数的方法自己尝试用普通子函数实现。求解这个问题的代码如下：

```
function InlineSubAnonymousNestedDemo
%% 用 inline 解决
tic;
k = linspace(0,5);
y1 = zeros(size(k));
for i = 1:length(k)
    kk = k(i);
    fun = inline(['sin(',num2str(kk),'*x).*x.^2']);
    y1(i) = quadl(fun,0,5);
end
time = toc;
disp(['用 inline 方法的时间是:',num2str(time),'秒！'])
%% 用 anonymous function 解决
tic;
f = @(k) quadl(@(x) sin(k.*x).*x.^2,0,5);
kk = linspace(0,5);
y2 = zeros(size(kk));
for ii = 1:length(kk)
    y2(ii) = f(kk(ii));
end
```

```
time = toc;
disp(['用 anonymous function 方法的时间是:',num2str(time),'秒！'])
%% 用 nested function 解决
    function y = ParaInteg(k)
        y = quadl(@(x) sin(k. * x). * x.^2 ,0,5);
    end
tic;
kk = linspace(0,5);
y3 = zeros(size(kk));
for ii = 1:length(kk)
    y3(ii) = ParaInteg(kk(ii));
end
time = toc;
disp(['用 nested function 方法的时间是:',num2str(time),'秒！'])
%% 用 arrayfun + anonymous function 解决
tic;y4 = arrayfun(@(k) quadl(@(x) sin(k. * x). * x.^2,0,5),linspace(0,5));time = toc;
disp(['用 arrayfun + anonymous function 方法的时间是:',num2str(time),'秒！'])
plot(kk,y2);
xlabel('k');
ylabel('f(k)')
end
```

运行这个程序，其结果如下：

```
用 inline 方法的时间是:0.96154 秒！
用 anonymous function 方法的时间是:0.13904 秒！
用 nested function 方法的时间是:0.1368 秒！
用 arrayfun + anonymous function 方法的时间是:0.13907 秒！
```

得到的图形如图 2.4 所示。

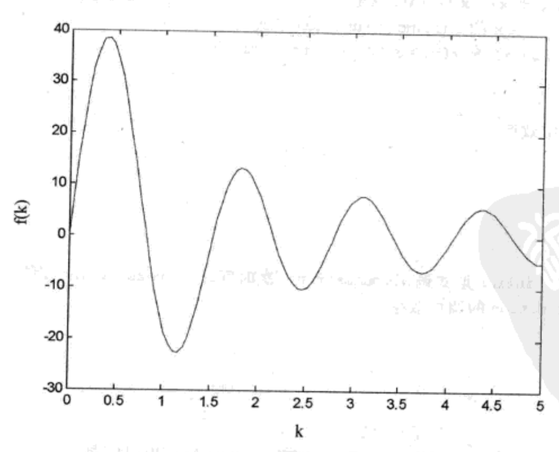


图 2.4　$f(k)=\int_0^5 \sin(kx)x^2\,dx$ 函数图像

从例2.3-2可以看出，匿名函数和嵌套函数的效率要明显高于inline函数，而且在参数传递方面要比inline函数方便高效。不熟悉匿名函数或者嵌套函数的读者可能对匿名函数或者嵌套函数的用法理解得不是很透彻，这没关系，对其的详细介绍将放在后续章节进行，通过本例读者只要能建立起对匿名函数和嵌套函数的初步印象即可。本例中还有用arrayfun的示例，arrayfun最早出现在MATLAB 7.1版中，其功能是将一特定函数（用户指定函数句柄）应用到一数组的每个元素上，返回值是和输入数组尺寸一样大的数组，返回数组和输入数组对应元素是相应的函数输出。这里的数组不仅包括数值型的还可以是元胞（cell）型和结构（struct）型的。利用arrayfun，可以简写很多循环的代码，后面章节有对其专门的介绍。类似的函数还有structfun、cellfun、spfun等。

这里需要强调的是，inline对象作为MATLAB 7.0之前构造简短函数常用的结构，在7.0之后的版本中已经过时了，取而代之的是匿名函数（anonymous function），inline函数能实现的，匿名函数完全可以更好地实现，并且调用效率要比inline函数高得多。而且用匿名函数很多可以方便实现的功能，用inline函数却很难方便实现。

这里先粗略比较一下MATLAB中各种函数结构的调用效率，说明选择好函数结构的重要性。下面以 $f(x)=x$ 这个简单的函数为例，来比较内联（inline）函数、匿名（anonymous）函数、嵌套（nested）函数、普通子函数（sub-function）的调用效率。

【例2.3-3】 编写 $f(x)=x$ 相应的内联函数、匿名函数、嵌套函数、普通子函数形式代码，并分别调用10 000次，比较各种函数结构调用时间的差异。

比较的代码如下：

```
function InlineSubAnonymousNestedCallDemo
% 各种类型函数调用效率比较
n = 10000;
f1 = inline('x'); % f(x) = x 的 inline 形式
f2 = @(x) x; % f(x) = x 的 anonymous function 形式
    function f3 = f3(x)  % f(x) = x 的 nested function 形式
        f3 = x;
    end
%% inline 的调用效率
tic
for k = 1:n
    f1(1);
end
time = toc;
disp(['f(x) = x 的 inline 形式调用',num2str(n),'次时间是:',num2str(time),'秒！'])
%% anonymous function 的调用效率
tic
for k = 1:n
    f2(1);
end
time = toc;
disp(['f(x) = x 的 anonymous function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒！'])
```

```
%% nested function 的调用效率
tic
for k = 1:n
    f3(1);
end
time = toc;
disp(['f(x) = x 的 nested function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒! '])
%% sub - function 的调用效率
tic
for k = 1:n
    f4(1);
end
time = toc;
disp(['f(x) = x 的 sub - function 形式调用',num2str(n),'次时间是:',...
    num2str(time),'秒! '])
end

function f4 = f4(x) % f(x) = x 的 sub - function 形式
f4 = x;
end
```

运行结果如下：

```
f(x) = x 的 inline 形式调用 10000 次时间是:1.1487 秒!
f(x) = x 的 anonymous function 形式调用 10000 次时间是:0.0060664 秒!
f(x) = x 的 nested function 形式调用 10000 次时间是:0.0042187 秒!
f(x) = x 的 sub - function 形式调用 10000 次时间是:0.0034907 秒!
```

由此可以看出，匿名函数、嵌套函数和普通子函数的调用效率基本上相差不太大，而内联函数的调用效率是上述几种函数类型调用效率的几百分之一！

因此在使用高版本 MATLAB 编程的过程中，要坚决摒弃 inline 这种过时的函数结构，而要根据需要选用匿名函数或者嵌套函数以及子函数。一般说来，简短的数学表达式函数或者可以写成一行代码的函数可以选用匿名函数来实现；而复杂的，内部要做较多事情的函数可以选用子函数和嵌套函数来实现；其中在变量共享以及传递方面，嵌套函数具有较大的优势，尤其是用子函数实现并且用到全局变量的场合，用嵌套函数可以避免用全局变量，并且实现起来方便容易。后续专门讨论嵌套函数的章节会详细介绍嵌套函数的用法。

2.3.3　选用恰当的数据类型

在 2.3.2 节讨论了不同的函数结构实现相同功能之间的效率差异。MATLAB 同样有多种数据类型，不同的数据类型在存储和访问效率上也不尽相同，选用恰当的数据类型对写出高效的 MATLAB 代码同样具有重要的意义。下面就最常用的三种数据类型：double 型数组、cell 数组、struct 数组，讨论其访问效率。看下面的例子：

【例 2.3-4】　比较 double 型数组、cell 数组和 struct 数组的访问效率。

```
function VisitSpeedOfDifferentDataTypes
n = 30000;
a = 8;
b{1} = 8;
c.data = 8;
%%
tic;
for k = 1:n
    a;
end
time = toc;
disp(['访问',num2str(n),'次 double 型数组时间是:',num2str(time),'秒！'])
%%
tic;
for k = 1:n
    b{1};
end
time = toc;
disp(['访问',num2str(n),'次 cell 型数组时间是:',num2str(time),'秒！'])
%%
tic;
for k = 1:n
    c.data;
end
time = toc;
disp(['访问',num2str(n),'次结构数组时间是:',num2str(time),'秒！'])
```

运行结果如下：

```
访问 30000 次 double 型数组时间是:0.00010113 秒!
访问 30000 次 cell 型数组时间是:0.0069649 秒!
访问 30000 次结构数组时间是:0.00011091 秒!
```

由此可以看出，就访问效率来说，double 型数组和 struct 数组的访问效率相当，大约都是 cell 型数组的 70 倍。

就在内存中存储来说，即使空的 cell 型数组，每个 cell 单元也要占用 4 B 的内存空间。这是因为，cell 型的数组不需要连续内存空间来存储，因此，每个 cell 单元需要记录其在内存中位置等一些信息，每个 cell 单元无论空与否，都在内存中占有固定的 4 B 用来存储这些信息。而一个 1 * 1 的 struct 数组，每一个域无论空与否，都会占用 124 B 的内存空间。

综合占用内存大小，以及访问效率等因素，在设计程序时应尽量采用 double 型数组。

选择好数据类型后，我们还要注意高效的访问方法。就 double 型数组来说，由于 MATLAB 是按列顺序存储数据的，因此按列访问、遍历数值数组要比按行访问、遍历快。看下面的例子：

```
function ColLoopVsRowLoop
n = 3000;
a = rand(n);
tic;
```

```
for ii = 1:n
    for jj = 1:n
        a(ii,jj);
    end
end
toc

 tic;
for ii = 1:n
    for jj = 1:n
        a(jj,ii);
    end
end
toc
```

运行结果如下：

```
Elapsed time is 0.136464 seconds.
Elapsed time is 0.069589 seconds.
```

可见按列遍历比按行快一倍左右。

2.3.4　减少无谓损耗——给一些函数“瘦身”

编程者要清楚，在 MATLAB 程序完成一个特定任务时，哪些是完成这个特定任务必须执行的代码，哪些是不必须的。举例来说，MATLAB 自身以及工具箱里的很多函数具有较广的通用性，所谓通用性是添加必要的代码能顾及到一个个具体的情形，这样对于每个具体的情形，代码实际上都有冗余。如果我们的程序某些部分对通用性要求不高，那么在优化程序时要考虑将不必要的代码去掉，这样的结果就是重写一些简单的函数，轻装上阵，举例如下：

【例 2.3-5】 median 函数的“瘦身”。

有兴趣的读者可以在命令窗口中运行“edit median”或者“type median”看一看 MATLAB 自带的“median”函数的源代码。

读者会发现，“median”函数源代码比较长，而且为了保证函数具有较广的通用性，源代码里各种分支判断和类型判断较多。这对于特定的某个求中位数的函数来说，可能很多代码都是不必要执行的，因为求中位数，最核心的是排序程序段，而且求中位数也不难自己实现，因此针对特定问题我们可以自己写“瘦身的 median 函数”代码。譬如程序中可能会频繁求一个维数为 1 000 左右的向量的中位数，可以写如下的函数：

```
function b = mymedian(a)
a = sort(a);
n = length(a);
half = floor(n/2); % 若非整数则向下取整,整数维持不变
b = a(half + 1);
if half * 2 == n % 若 n 为偶数,b 等于排好序的中间两个数平均值
    b = (b + a(half))/2;
end
```

做如下测试，测试和运行结果如下：

```
>> clear
a = rand(1000,1);
tic;for k = 1:10000;c = median(a);end;toc
tic;for k = 1:10000;d = mymedian(a);end;toc
Elapsed time is 0.859531 seconds.
Elapsed time is 0.690383 seconds.
>> isequal(c,d)
ans =
     1
```

可见“瘦身”后的 mymedian 函数运行速度提高了 20%左右。这是因为“瘦身”后的mymedian函数考虑的情况及其分支较 median 函数少。需要说明的是，当 a 的长度很大时，执行分支判断语句的时间远小于排序本身的时间，median 和 mymedian 的时间差距不会太明显。但是当 a 的长度较小时，执行排序的时间小于执行额外语句的时间时，这种差距就明显了。同样上述代码，不同的是 a 的长度是 100 了，两个函数运行时间差距就大了，测试和运行结果如下：

```
>> a = rand(100,1);
tic;for k = 1:10000;c = median(a);end;toc
tic;for k = 1:10000;d = mymedian(a);end;toc
Elapsed time is 0.228408 seconds.
Elapsed time is 0.060108 seconds.
>> isequal(c,d)
ans =
     1
```

这时候瘦身的函数所用时间仅为原始 median 的 1/4 多点。

本例的启示是在通用性要求不高的场合，可以根据我们的需要来给 MATLAB 函数瘦身，当然要权衡瘦身的难易程度和获得的效率提升程度。

2.3.5 变“勤拿少取”为“少拿多取”

通过前面章节的介绍，我们可以了解如下事实：

在 MATLAB 中，函数调用都要有一定的开销，这种开销一般是 built-in（像 sin、cos、zeros、ones、find、all、any 等，但凡在命令窗口运行“type 文件名”，MATLAB 会显示“XXX is a built-in function.”这样的函数）为最低，其次是一般的 M 文件，即工具箱里能看到源代码的那些 M 函数以及用户自己写的 M 文件、子函数、nested function、匿名函数等。上面这些函数类型总的来说调用开销都比较小，而调用开销最大的是 inline 函数类型（在高版本的 MATLAB 中应当摒弃 inline 这种落后的函数类型）。函数调用开销相对于一些简单的计算来说，往往是相当可观的，即使是使用 built-in 函数类型。

另外我们还注意到为了使得函数具有通用性，很多函数刚开始都会有一些判断、准备工作，往往判断好几层才到了真正算法实现部分。在 2.3.4 节，讨论了“函数瘦身”所带来的运行速度上的提升。可是 MATLAB 作为方便易用的科学计算语言，其优势之一就是集中了大量现成的功能强大的函数，如果所用 MATLAB 自带函数的核心算法实现较复杂，那么为“函数瘦身”就可能得不偿失了。这时候为了尽可能多地降低函数调用而产生的无谓的损耗，就要想

尽办法把需要多次函数调用才能完成的工作用较少次数的函数调用来完成。我们形象地称之为：变"勤拿少取"为"少拿多取"。下面以一个例子来说明：

【例 2.3-6】 数组 **A** 是一个 7×1000000 的矩阵，每行的形式类似下面所示：

A=[100　2　6　96　8　7　20；

15　69　7　6　4　20　11；

21　101　45　48　6　5　4；

...]；

b 是一维数组[4　6　8]；

试找出数组 **A** 中的每一行与数组 ***b*** 的交集，并将 ***b*** 中的交集元素置零，并将置零后的 ***b*** 存到 **B** 中。**B** 的形式为[4　0　0；0　0　8；0　0　8；…]。

本例如果按照常规思路做的话，可能会想到下面的办法：

```
function example2_3_6slow
A = unidrnd(100,1000000,7); % 随机生成 1000000 * 7 的 A 矩阵,A 的元素属于 1 到 100 的整数
B = zeros(1000000,3);
for m = 1:1000000
    a = A(m,:);
    b = [4 6 8];
    for ii = 1:3
        dd = a(a == b(ii)); % dd:a 中等于 b(ii)的元素
        if isempty(dd) == 0 % dd 不为空
            b(ii) = 0;
        end
    end
    B(m,:) = b;
end
```

仔细分析上面的代码会发现，需要遍历 **A** 的每一行，判断 **A** 的每一行是否有元素等于 ***b*** 的某个元素，如果有的话，就把相应的 ***b*** 的这个元素置为 0。这样对本例来说，光 isempty 就调用了 300 万次，可以想象，效率比较低。如果采用"少拿多取"的思想，可以做如下改进：

```
function example2_3_6fast1
clear
A = unidrnd(100,1000000,7); % 这里先假设 A 是一个随机矩阵
B = repmat([4,6,8],1000000,1);
tic;C = [any(ismember(A,4),2) any(ismember(A,6),2) any(ismember(A,8),2)];
B(C) = 0;
toc
```

或者改进如下：

```
function example2_3_6fast2
clear
A = unidrnd(100,1000000,7);
B = repmat([4,6,8],1000000,1);
tic;C = [any(AA == 4,2) any(AA == 6,2) any(AA == 8,2)];
B(C) = 0;
toc
```

上述两个程序的共同之处在于：一次处理很多数据。这样，原本要调用300万次的“比较”操作，现在只调用三次，只不过每次需处理整个 **A** 矩阵。然后结合 any 函数给出最后要被置零的逻辑索引。

上面代码不难看懂，关键是上面叙述的编程理念。

2.3.6 循环注意事项

很多人编写循环时，往往忽略很重要的事实，那就是同样循环层数、次数，不同的写法可能造成速度巨大的差异。编写循环时应该遵循两个重要的原则：

1) 按列优先循环(因为 MATLAB 按列循序存储矩阵)；

2) 循环次数多的变量安排在内层。

我们以具体例子来说明上述原则的重要性。请看示例：

【例 2.3-7】 对于 n×n 的矩阵 **A**，生成一个矩阵 **B**，使得其元素满足下列关系式：

$$B(i,j)=\Big(\sum_{\substack{i-1\leqslant k\leqslant i+1\\ j-1\leqslant l\leqslant j+1}} A(k,l)\Big)/9,\quad 2\leqslant i\leqslant n-1,\quad 2\leqslant j\leqslant n-1$$

对于本例，可以用多种方法来求解。由于现在是在讨论循环注意事项，因此用循环的方法来处理，而且是用四重循环。值得注意的是，用这么多层循环，在老版本时代，历来是被强烈反对的。我们看看在 MATLAB R2009a 平台下，不同的写法效率能有多大差距。读者可以用二重循环或者想尽办法避免循环试一试，并和效率高的四重循环做个对比(当然平台是 MATLAB R2009a)。下面以一个 512×512 的随机矩阵 **A** 为例来说明。

先看方法一：

```
function y = ForLooPCompare1(x)
y(512,512) = 0;
tic;
for i = 2:511
    for j = 2:511
        for k1 = -1:1
            for k2 = -1:1
                y(i,j) = y(i,j) + x(i + k1,j + k2)/9;
            end
        end
    end
end
toc
```

本方法将循环次数少的放到了内循环，外循环依次是 i，j，并且是按行优先循环。

再看第二种方法：

```
function y = ForLooPCompare2(x)
y(512,512) = 0;
tic;
for j = 2:511
    for i = 2:511
        for k2 = -1:1
```

```
            for k1 = -1:1
                y(i,j) = y(i,j) + x(i+k1,j+k2)/9;
            end
        end
    end
end
toc
```

本方法同样将循环次数少的放到了内循环，不同的是内外循环都是列优先。

第三种方法：

```
function y = ForLooPCompare3(x)
y(512,512) = 0;
tic;
for k1 = -1:1
    for k2 = -1:1
        for i = 2:511
            for j = 2:511
                y(i,j) = y(i,j) + x(i+k1,j+k2)/9;
            end
        end
    end
end
toc
```

本方法将循环次数少的放到了外循环，内外循环都是按行优先循环。

第四种方法：

```
function y = ForLooPCompare4(x)
y(512,512) = 0;
tic;
for k2 = -1:1
    for k1 = -1:1
        for j = 2:511
            for i = 2:511
                y(i,j) = y(i,j) + x(i+k1,j+k2)/9;
            end
        end
    end
end
toc
```

该方法不仅将循环次数少的放到了外循环，而且内外都是按照列优先的循环顺序。下面是这四种方法运行效率对比：

```
x = rand(512);
y1 = ForLooPCompare1(x);
y2 = ForLooPCompare2(x);
y3 = ForLooPCompare3(x);
```

```
y4 = ForLooPCompare4(x);
Elapsed time is 0.330648 seconds.
Elapsed time is 0.281970 seconds.
Elapsed time is 0.322625 seconds.
Elapsed time is 0.057423 seconds.
```

由此可见，同时遵循“按列优先循环”、“循环次数多的变量安排在内层”两个原则的方法耗时最少。

2.3.7 逻辑索引和逻辑运算的应用

我们知道，MATLAB的矩阵元素索引有两类：一类是数值索引，另一类是逻辑索引。前者又可分为线性索引(linear index)和下标索引(subscripts)。就运行效率来说，逻辑索引要高于数值索引，所以访问矩阵元素能用逻辑索引的就用逻辑索引。

这里需要强调的是find函数返回的是数值索引，因此，如果不需要对返回的索引值做进一步的操作，只是单纯寻找满足条件的一些元素的话，find函数完全可以不用。看下面的例子：

【例2.3-8】 随机生成一个1000×1000的矩阵**A**，**A**的元素服从[0,1]区间均匀分布，并找出(0.3,0.7)区间的所有元素。

请看用数值索引和逻辑索引的运行效率对比：

```
A = rand(1000);
tic;B = A(find(A > 0.3 & A < 0.7));toc
tic;C = A((A > 0.3 & A < 0.7));toc
Elapsed time is 0.056147 seconds.
Elapsed time is 0.047591 seconds.
```

关于逻辑运算，MATLAB提供了一系列内置的逻辑运算函数，这些函数运行效率往往比较高，可以根据需要进行选用。这些函数通过各自函数帮助文档页面下面的see also选项里列出的函数彼此连接起来，这样可以减少使用时候的记忆负担。譬如我们运行doc any，帮助文档里除了列出any函数的使用说明外，在下面的see also选项里还列出和any一类的其他函数，这些函数的帮助文档下面又列出和自身关系密切的其他函数。

此外，在命令窗口中运行“doc is”可以列出一系列有关判断的逻辑函数。这些函数的说明文档都比较容易明白，由于篇幅关系，这里对每个函数的功能就不做介绍了，读者可以看看这些函数的帮助文档。

2.4 应用高版本向量化函数提高开发效率

从MATLAB 7.0开始，陆续增加了一些向量化函数，使用这些函数可以减少很多循环的使用，在保证代码运行效率的前提下，使代码更加简洁。需要说明的是，很多情况下，使用这些函数其运行效率并不比恰当使用循环快多少，使用它们主要是为了提高开发效率，使代码更加简洁。下面逐一介绍这些函数。

2.4.1 accumarray函数

accumarray函数最早出现于7.0版(R14)，在随后的7.1 (R14SP3)，7.2 (R2006a)版里

又对其功能进行了增强。

常用的 accumarray 函数的调用格式有以下几种：

```
A = accumarray(subs,val)
A = accumarray(subs,val,sz)
A = accumarray(subs,val,sz,fun)
A = accumarray(subs,val,sz,fun,fillval)
A = accumarray(subs,val,sz,fun,fillval,issparse)
```

下面以 accumarray 函数最全的参数调用格式 A = accumarray(subs, val, sz, fun, fillval, issparse)为例，说明一下 accumarray 函数最常用的使用方法。

给定一组索引值 subs，一般要求其是 n 行 m 列的正整数矩阵，val 是一长度为 n 的数组(行数组或者列数组)，subs 的每一行表示一个索引值，与 val 相应的元素对应。accumarray 函数的功能是将索引值相同的 val 分别分组，然后用事先指定的函数 fun 作用于每一组，输出的结果是一个矩阵 **A**，矩阵 **A** 的维数等于 subs 的列数，

如果 **A** 的某个元素对应的索引值(即 **A** 通过该索引值可以访问该元素，譬如 **A**=[0.2 −5;7 8]，对于−5 来说，其索引值就是[1,2]，表示第一行第二列，**A**(1,2)可以访问到−5 不曾在 subs 里出现过，那么 **A** 的这个元素只能是 0(默认)，或者是我们事先指定的 fillval。如果出现过，则 **A** 的这个元素是相应的 fun 作用的 val 中对应元素的值。我们可以为 **A** 重新指定尺寸：sz，只是 sz 的每个分量都不小于对应的 subs 列中最大的值。issparse 参数是来说明是否以稀疏矩阵的形式输出 **A**。

accumarray 函数用法比较抽象，很多读者包括一些熟练使用 MATLAB 的朋友即使看了英文帮助文档，还是对其用法不甚明白。所以看完上述解释后如果还是一头雾水，那么不要紧，可以结合下面的例子来看。

下面看几个关于 accumarray 函数的例子：

【例 2.4－1】 生成一个 100 000×1 的 ***a***，其元素服从[1,200 000]之间离散均匀分布，找出都有哪些元素出现了。

先用下列语句生成 ***a***：

```
a = unidrnd(200000,100000,1);
```

下面有几种办法找到 ***a*** 中都有哪些元素出现了。如下(环境：MATLAB R2009a＋Windows XP sp2)：

方法 1：

```
tic;b = union(a,a);toc;
Elapsed time is 0.033577 seconds.
```

方法 2：

```
tic;c = unique(a);toc;
Elapsed time is 0.015064 seconds.
```

方法 3：

```
tic,d = accumarray(a,1,[200000,1]);e = find(d);toc
Elapsed time is 0.009958 seconds.
```

由于 MATLAB 第一次运行函数会有编译等额外工作，因此第一次运行较慢，上述运行时间都是取的第三次运行的时间。可以看出使用 accumarray 函数的方法 3，不仅得到了哪些元素出现了（即向量 ***e***），同时还给出了每个出现的元素出现的次数——向量 ***d***，所用时间也是最少的。

【例 2.4-2】 在 1000×1000 的正方形区域内随机生成 100 000 个点（坐标值是整数），统计每个坐标点上生成的点的个数。

在这个例子下，像例 2.4-1 一样简单应用 union 和 unique 就不行了。通常考虑用循环：

```
clear;
p = unidrnd(1000,100000,2);
tic;
A(1000,1000) = 0;
for  k = 1:100000
A(p(k,1),p(k,2)) = A(p(k,1),p(k,2)) + 1;
end
toc
Elapsed time is 0.020455 seconds.
```

而用 accumarray 函数可以这样：

```
tic;a = accumarray(p,1,[1000,1000]);toc
Elapsed time is 0.017606 seconds.
isequal(A,a)
ans =
     1
```

可见，用 accumarray 函数，仅需一句代码，既简洁又高效。

【例 2.4-3】 1 000 人，身高分布在 170～180 cm，体重在 110～200 斤，年龄分布在 20～50岁，计算身高体重都相等的人的年龄平均值。结果用矩阵来表示：行数表示身高，列数表示体重，矩阵元素表示年龄的平均值。

首先，生成数据：

```
rand('state',0)
height = unidrnd(10,1000,1) + 170; % 身高的数据
rand('state',0)
weight = unidrnd(90,1000,1) + 110; % 体重数据
rand('state',0)
old = unidrnd(30,1000,1) + 20;
```

利用 accumarray 函数计算的语句如下：

```
tic;mo = accumarray([height,weight],old,[],@mean);toc
Elapsed time is 0.002271 seconds.
```

这个矩阵比较稀疏，因此也可以用稀疏矩阵来表示结果：

```
tic;mo = accumarray([height,weight],old,[],@mean,0,true);toc
Elapsed time is 0.002072 seconds.
```

维数大时，稀疏矩阵会有优势，当然这个例子还不明显。读者有兴趣可以试试传统方法的效果。

以上仅举了三个例子，实际上，accumaary 函数的应用方法非常灵活，尤其在对于很多要操作大矩阵的情况下。建议读者可以仔细研究下 accumarray 函数的用法，并在实际当中注意应用。

2.4.2　arrayfun 函数

相信很多读者都有这样的体会，看循环代码的时候比较累人，尤其是多重嵌套循环。很多时候我们编写的函数的输入变量是标量，而又要对很多组参数进行函数调用，而函数是很难或者不支持向量化的，这在 MATLAB7 以前的版本中我们只能通过老老实实通过循环来实现。虽说高版本下恰当使用 MATLAB 的循环，效率已不是大的瓶颈，但如果有效率和循环相当或者更好，并且很简洁的其他方式，想必绝大多数读者是不会拒绝的。arrayfun 函数和后面小节的 cellfun，spfun，structfun 等函数就是这样的一类函数。本小节先讨论 arrayfun 函数的用法。

arrayfun 这个 built-in 函数实现的是将指定的函数应用到给定数组包括结构数组在内的所有元素。这样很多以前不可避免的循环现在可以向量化了。看下面的几个例子：

【例 2.4-4】　生成一个这样的 n×n 矩阵 ***a***：***a***(i,j)=dblquad(@(u,v) sin(u) * sqrt(v),0,i,0,j)，以 n=10 为例。

这个问题放在 MATLAB 7 以前，我们可能这样做：

```
a = zeros(10);
for ii = 1:10
    for jj = 1:10
        % 求 sin(u) * sqrt(v)的二重积分
        a(ii,jj) = dblquad(@(u,v) sin(u) * sqrt(v),0,ii,0,jj);
    end
end
```

现在只需这样：

```
[J,I] = meshgrid(1:10);
a1 = arrayfun(@(ii,jj) dblquad(@(u,v) sin(u) * sqrt(v),0,ii,0,jj),I,J);
```

【例 2.4-5】　验证角谷猜想，一个正整数 n，如果是偶数除以 2，如果是奇数乘以 3 加 1，得到的新数继续按上述规则运算，最后结果都为 1。验证 1～100 000 内的正整数。先编写单个数的验证函数：

```
function f = SizuoKakutani (n)
if n<1 || mod(n,1)~= 0
    errordlg('n 必须是正整数')
    f = n;
    return
end
while n>1
```

```
    if mod(n,2) == 1
        n = n * 3 + 1;
    elseif mod(n,2) == 0
        n = n/2;
    end
end
f = n;
```

复制代码验证 1:100000，以前我们可能这样做：

```
a = zeros(1,100000);
for k = 1:100000
a(k) = SizuoKakutani(k);
end
all(a)
```

现在只需这样：

```
all(arrayfun(@SizuoKakutani,1:100000))
```

可见，arrayfun 函数可以使代码简化很多。举这两个例子目的是为了说明很多以前不能简写的很多循环的程序利用这个函数可以在保证效率的前提下简化代码。关于 arrayfun 函数其他使用方法，有兴趣的读者可以运行 doc arrayfun，看其帮助文档加以了解。

2.4.3 bsxfun 函数

以前，当我们想对一个矩阵 **A** 的每一列或者每一行与同一个向量 ***a*** 进行某些操作(比较大小，乘除等)时，只能用循环方法或者利用 repmat 函数将要操作的向量 ***a*** 复制成和 **A** 一样尺寸的矩阵，进而进行操作。从 MATLAB R2007a 版本开始，再遇到类似的问题时，有了简洁高效的方法，即利用 bsxfun 函数。

为了节省篇幅，bsxfun 的调用格式就不列在这里了，读者可以参考帮助文档，下面给出两个 bsxfun 的例子，读者可以仔细揣摩，相信会从中得到使用 bsxfun 的很多启发。

【例 2.4-6】 有如下矩阵：

$$\boldsymbol{A}=\begin{pmatrix}1 & 2 & 3 & 1\\ 2 & 3 & 4 & 2\\ 3 & 3 & 8 & 3\end{pmatrix}$$

向量为 $\boldsymbol{b}=[1\quad 2\quad 3]^{T}$，请找出 $\boldsymbol{b}$ 在 $\boldsymbol{A}$ 矩阵列中的位置 loc=[1,4]。

下面给出五种方法，这五种方法每种方法都只有一句话，包括 bsxfun 的使用和前面 arrayfun函数的使用。读者可以仔细体会其用法，相信会受益匪浅。

方法 1：

```
loc = find(all(bsxfun(@eq,A,b))) % 把 A 的每一列和 b 用 == (@eq)来判断，找出全 1 的列
```

方法 2：

```
loc = find(arrayfun(@(n) all(A(:,n) == b),1:4)) % 用 arrayfun 对 n 进行 1:4 的遍历
```

方法 3：

```
loc = find(all(~bsxfun(@minus,A,b))) % 把 A 的每一列和 b 来相减(@minus),求反后找出全 1 的列
```

方法 4：

```
loc = find(arrayfun(@(n) isequal(A(:,n),b) ,1:4)) % 用 arrayfun 对 n 进行 1:4 的遍历后用
isequal 函数来判断 A 的每列和 b 是否相等
```

方法 5：

```
loc = find(b' * A == sum(b.^2)) % b 的转置和 A 相乘,然后和 b.^2 每列的和进行比较,找到相等的
```

上述几种方法都可以达到目的，但是思路各不相同。读者可以用较大规模的矩阵比较上述各方法的效率。

再看一个例子：

【例 2.4-7】 如何将一个矩阵的每行或每列元素分别扩大不同的倍数？如[1　2　3;4　5　6;7　8　9]，第一列元素乘以 1，第二列元素乘以 2，第三列元素乘以 4。

利用 bsxfun 函数，可以给出下列代码：

```
a = [1,2,3;4,5,6;7,8,9];
acol = bsxfun(@times,a,[1 2 4])
```

事实上，bsxfun 的第一个输入参数是函数句柄，其支持的函数类型有多种，帮助文档里面都列了出来，大家在使用的时候可以参考帮助文档。

从 MATLAB R2009b 开始，bsxfun 函数开始更好地支持多线程了，即不需要用户干预，其内部已经较之前的版本更好的支持多线程了。更多的较之以前更好地支持多线程的函数还有：sort（有其对于大矩阵），mldivide（对于稀疏矩阵），qr（对于稀疏矩阵），filter，gamma，gammaln，erf，erfc，erfcx，erfinv 等函数。

2.4.4　cellfun 函数

类似 arrayfun 函数，cellfun 函数是将指定的函数应用到元胞数组内每个元胞存储的内容上。看下面的简单例子：

【例 2.4-8】 A={'Hello','MATLAB','I love MATLAB','MATLAB is powerful','MATLAB is the language of technical computing'}；试统计 A 中每个元胞单元存放的字符串的长度。

本例用 cellfun 的代码非常简单，只一句话即可完成任务。

```
len = cellfun(@length,A)
len =
     5     6    13    18    45
```

【例 2.4-8 续】 试提取 A 中每个元胞内容前三个字符，并全部转化成大写。

用 cellfun 做同样很简单：

```
Upp = cellfun(@(x) upper(x(1:3)),A,'UniformOutput',false)
Upp =
    'HEL'   'MAT'   'I L'   'MAT'   'MAT'
```

这里要说明一点。cellfun 和 arrayfun 函数都可作用于一个 cell 数组(arrayfun 函数还可以用于数值，结构等数组)，那它们有什么区别呢？在回答这个问题前，我们先打个比方：

cell 数组好比一栋"楼房"，不同的行数好比不同的"楼房"层数，每个 cell 单元好比楼层上的"房间"，而每个 cell 单元里面具体存放的东西好比是放在"房间"里的东西。

cellfun 的输入函数操作的是放在"房间"里的东西。而 arrayfun 的输入函数操作的是具体的每个"房间"整体。读者可以体会下列代码的运行结果。

```
a = {12,34,55,66,77}
a =
    [12]    [34]    [55]    [66]    [77]
>> b = cellfun(@(x) x,a)
b =
    12    34    55    66    77
>> c = arrayfun(@(x) x,a)
c =
    [12]    [34]    [55]    [66]    [77]
>> isequal(c,a)
ans =
     1
```

cellfun 和 arrayfun 函数的指定函数都相同，都是 $f(x)=x$ 这个简单的函数，可以清楚看出，cellfun 函数把每个"房间"里的数都取了出来，由于指定函数是 $f(x)=x$，所以没有对其进行任何操作，只是将它们放到一个和 a 行列数相同的数值数组 b 里。而 arrayfun 函数则是将每个"房间"连同里面的东西都取了出来，所有都取完后再拼成一个整体 c，因此 c 还是等于 a。

如果 cellfun 函数的指定函数操作每个单元返回的值无法用数值数组来统一表示，那么必须用 cell 型数组来表示。譬如当 a={1,2,'abc'}时，b=cellfun(@(x) x ,a)这么写就会报错了，为什么？因为其指定函数 $f(x)=x$ 操作每个单元返回的值不是同一类型，无法用数值矩阵统一表示，只能用 cell 数组表示。于是应该写为 b=cellfun(@(x) x ,a,'UniformOutput',false)；这时候 b 和 c=arrayfun(@(x) x ,a)是相同的。

2.4.5 spfun 函数

spfun 函数的使用方法比较简单，即将指定的函数 fun 应用到稀疏矩阵 $\boldsymbol{a}$ 每个不为 0 的元素上，返回一个和 $\boldsymbol{a}$ 尺度相同的，且具有稀疏结构的矩阵。看下面的例子。

【例 2.4-9】 演示 spfun 函数的简单使用方法：

```
a = sparse([1 3 20 60 100],[2 20 30 60 80],[1 2 3 4 5])
a =
     (1,2)          1
     (3,20)         2
    (20,30)         3
    (60,60)         4
   (100,80)         5
>> sa = spfun(@(x) x.^2 + 1,a)
sa =
     (1,2)          2
     (3,20)         5
    (20,30)        10
    (60,60)        17
   (100,80)        26
```

2.4.6 structfun 函数

structfun 函数用法和上述介绍的函数类似，不过 structfun 函数要求操作的结构数组是标量结构数组，所谓标量结构数组是指在 MATLAB 工作空间的“变量 value”那一栏显示为 1×1 struct的结构数组。

structfun 所接收的函数将作用到所接收的结构的各个域上。看下面的例子：

```
s.f1 = magic(3);
s.f2 = magic(5);
s.f3 = magic(8);
SumFields = structfun(@(x) sum(x) , s,'UniformOutput', false)
SumFields =
     f1: [15 15 15]
     f2: [65 65 65 65 65]
     f3: [260 260 260 260 260 260 260 260]
```

从上面的例子可以看出，新的结构数组 SumFields 的域和 strucfun 接收的结构数组 s 的域相同，每个域的内容是 sum 函数作用于 s 的相应域的内容的结果。

第3章 MATLAB处理海量数据

相信很多使用过MATLAB的读者都遇到"out of memory"的错误提示。出现这样的提示说明MATLAB的工作空间中的变量占据的内存空间总和超出了当前可用内存空间的总和。在不增加物理内存条件下,该怎么办?有的时候需要处理大量的数据,怎么样才能高效处理?

我们还经常听到MATLAB使用者抱怨MATLAB的内存管理机制很差,本来机器4 GB的物理内存,可是最多只能开1 GB大小的矩阵。怎么样才能有效利用4 GB的内存呢?

类似的这些问题是这章的主要讨论问题。

3.1 处理海量数据时遇到的问题

3.1.1 什么是海量数据

海量数据,顾名思义,数据量一般大得多。海量数据一般是相对当前计算机处理能力而言的,今天看来是海量数据,也许随着计算机硬件能力的提升在不久的将来就不算海量数据了。即使是同一时期,不同的应用领域,不同的学科以及不同的硬件条件下,海量数据所呈现的数据量也不尽相同。

对于用户个人来说,凡是在个人计算机上处理数据时经常出现"out of memory"提示的数据量,都可称之为海量数据。

3.1.2 经常遇到的问题

在有限的计算机处理能力下,我们处理海量数据经常遇到一些问题。这些问题根据表象可以分为两大类:一类是内存溢出(out of memory),一类是运行极其缓慢。

(1) 导致内存溢出的主要原因

1) 变量需要的存储空间超过了可用的内存空间。这是最简单的情况。原因是将要生成的矩阵太大,所需存储空间超过了所有可用的内存总和。但是很多时候矩阵需要的存储空间大小并没有超过可用内存总和,但同样有内存溢出的错误提示,其原因见下面第2)条。

2) 数值矩阵所需要的存储空间,超过了内存中最大的可用连续存储空间。MATLAB里默认用double型来表示数值变量,这样一个数字就需要8 B的存储空间。而MATLAB中的数值矩阵存储需要内存中连续的存储空间。这样如果内存中碎片过多,虽然可用的内存总和较大,但是最大的连续内存空间不大,这样当矩阵的元素较多时,就可能没有足够的连续内存空间来存储,从而发生内存溢出的错误提示。

譬如a=rand(10000,10000);生成10^8个(0,1)之间随机分布的随机数,需要的存储空间是8×10^8B,大约需要800 MB的连续内存空间。如果内存中最大的连续空间都小于$8\times$

10^8B,那么无论所有可用的内存空间之和是否大于 8×10^8B,MATLAB 都会给出"out of memory"错误提示的。

3) 程序设计方面考虑的不周,导致内存溢出。这方面的原因一般表现得比较隐蔽。我们在设计一些变量较多的复杂程序时,可能没有及时释放掉一些不再会用到的变量所占用的内存。这样随着程序的运行,变量越来越多,内存中碎片也越来越多,再要产生新的较大的矩阵,就可能发生内存溢出的问题。

4) 问题求解方法考虑欠周,导致内存溢出。同样的问题,可能由多种求解方法。不同方法的效率以及所占用的资源相差很大。问题规模小的时候,不用考虑这方面的问题,可是问题规模较大的时候就必须考虑具体的求解方法了,否则就可能会由于内存溢出而无法求解问题。

譬如符号计算。符号计算由于其语法简单,并且和传统教科书解决问题思路一致等特点,使得许多初学者很爱使用符号计算。但是符号计算由于精确性,使得其在计算过程中要保留大量的中间结果,这样导致复杂的计算问题用符号计算很容易内存溢出,即使不溢出,对于很多复杂的计算问题,符号计算也无法给出解析解。

还有当求解涉及较大规模的稀疏型矩阵的问题时,没有采用稀疏矩阵数据结构也容易导致内存溢出。

(2) 运行极其缓慢的原因

这是因为大数据量的处理总是伴随着大量的计算还有频繁的对内存各地址进行访问,这些开销往往比较占用资源。如果物理内存不够,系统开始调用页面文件来供程序使用,会使得程序运行速度有很大程度的下降。

本章后半部分将介绍不增加物理内存的情况下,可供 MATLAB 高效利用内存的方法。

3.2 有效设置增加可用内存

随着计算机硬件的发展,如今的内存价格较之以前低廉了很多,主流 PC 包括笔记本式计算机的内存配置基本都不低于 4 GB 了,这已经达到 32 位操作系统的寻址极限了。虽然 64 位操作系统可以支持更大的内存,但由于诸多因素导致 64 位操作系统完全取代 32 位操作系统还需要一段时间,在可预见的一段时间内,使用 32 位操作系统的读者还会很多,因此在 32 位操作系统下如何有效利用内存是这节主要讨论的内容。

本小节以 Windows XP 32 位操作系统为例来说明如何设置 boot. ini,使得 MATLAB 程序可以利用更多的内存。

3.2.1 系统默认下内存分配情况

首先需要澄清一个概念——虚拟内存(virtual memory)。有很多朋友习惯把硬盘上预留出来用做内存交换和扩展内存寻址空间的"交换文件"(也叫页面文件 Pagefile. sys)当做"虚拟内存"。因为它不是内存芯片,却在必要的时候和物理内存进行内存页交换,所以认为是"虚拟"的内存。当然这样讲有一定道理,也可以认为是人们通常说的"虚拟内存"代表的意思。但是有的时候,"虚拟内存"还代表另一种含义。简单说来,这种含义就是特定操作系统下多路程序进程共享的计算机物理内存以及页面文件。

Windows 的虚拟内存系统可以让所有的应用程序都运行在自己独占的私有所谓"虚拟地

址空间”里，这个“虚拟地址空间”对于应用程序的开发者以及使用者来说，使用起来就像单一任务操作系统 DOS 下独自访问实际物理地址一样，完全不需要考虑内存的调度、共享、安全等。而这些调度、共享、安全等的工作，则完全交由 Windows 虚拟内存系统来完成。

默认情况下，Windows 可以对总计4 GB的虚拟内存空间进行寻址。此地址空间中的2 GB为内核(操作系统)保留，另外2 GB是为用户程序保留的，值得注意的是每个进程都可以寻址这么多空间。实际运行中，OS(操作系统) 负责把进程提交的虚拟内存按页(一页 4 KB，工作集)映射到物理内存的实际页帧上(驻留集)。如果设置了硬盘上的交换文件，那么 OS 也会在物理内存和交换文件之间交换内存页，一些不活动的进程，它的虚拟内存页会被从物理内存中交换到硬盘上的页面文件中，等它活动的时候再交换回来。

有了上述讨论，下面可以进行本节下面的内容。打开 MATLAB，在命令窗口运行如下命令 feature memstats 或者 system_dependent memstats，(以上两个命令属于 MATLAB 中未公开的命令，即可以运行，但是帮助文档中没有)。我们可以得到当前 MATLAB 所运行的内存环境的一些参数值，如图 3.1 所示。

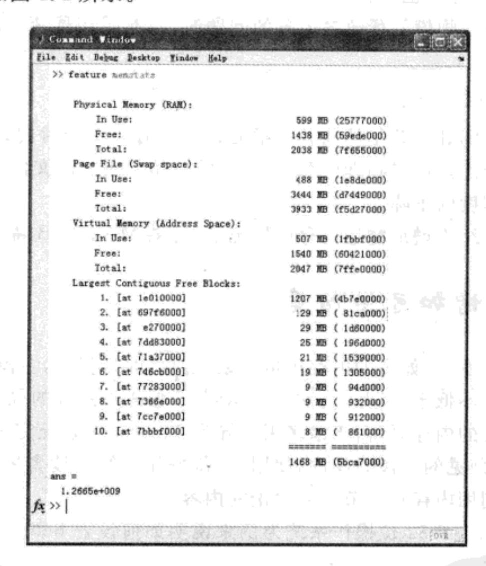


图 3.1　未公开 3 GB 模式 MATLAB 运行内存环境参数

上面显示的是 Windows XP 默认情况下，也就是没有打开 3 GB 开关情况下，笔者计算机上 MATLAB 运行的内存环境参数。

笔者计算机的物理内存是 2 GB，从图 3.1 Physical Memory 中可以看出来。Page File(简写为 PF)中 In Use 数值是和 Windows 任务管理器中 PF 使用率是一致的，也即“认可用量”中的“总数”或者任务管理器下面列的“提交更改”左边数值；而 Total 是和“认可用量”中的“限制”保持一致的，或者是“提交更改”右边数值。

Virtual Memory 那一项列的是 MATLAB 整个程序运行的虚拟内存空间，可以看出，在没有打开 3 GB 开关的情况下，MATLAB 寻址空间最多只有 2 GB。

Largest Contiguous Free Blocks 里列出了 10 个地址连续的最大的虚拟内存空间。由于 MATLAB 中数值矩阵的存放必须要地址连续的内存空间，因此，目前可以生成的最大矩阵或者向量含有的元素个数(以默认的 double 类型为例)是 1 207×1 024×1 024/8＝158 203 904 个，大约是 12 578×12 577 大小的一个矩阵。虽然这个矩阵是上述状态下可以生成的最大的数值矩阵，但是生成完这样一个矩阵后，MATLAB 还可以再生成其他的矩阵，只不过再生成的矩阵的最大尺寸取决于生成完第一个矩阵后，最大的连续内存空间大小。由此可以想象，如果内存碎片较多，会导致 MATLAB 能够处理的单个矩阵的最大尺寸大大减小。

3.2.2 打开 Windows 3 GB 开关

可以设置 boot. int 文件，使得 Windows 的 3 GB 开关打开，这样，MATLAB 的寻址空间就可以提高 1 GB。

可以按照下列步骤打开 3 GB 开关：

1) 右击“我的电脑”图标，选择“属性”命令，在弹出的“系统属性”界面中单击“高级”选项卡。

2) “在启动和故障恢复”一栏里单击“设置”按钮。

3) 在“启动和故障恢复”界面单击“编辑”按钮，弹出 boot. ini 记事本窗口，整个过程如图 3.2 所示。

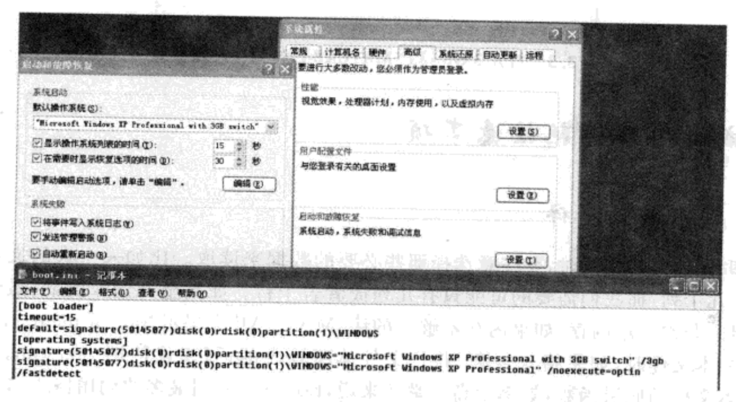


图 3.2 开启 Windows 3 GB 模式步骤图

4) 在 boot. ini 文件中的[operating systems]项里根据当前操作系统描述行增加一个新的操作系统描述行(带有“3GB switch”的那一行)，具体操作方法参照图 3.2 所示窗口中 boot. ini 所示。不同的系统描述行稍有出入，但方法是类似的。

5) 连续单击“保存”、“确定”按钮，关闭这些页面。计算机重启时会发现增加了一个开启 3 GB模式的系统选择项，选择它就进入了开启了 3 GB 的操作系统。

这时再打开 MATLAB，在命令窗口运行如下命令：feature memstats 或者 system_dependent memstats，会发现开启 3 GB 模式后 MATLAB 所运行的内存环境的参数值，如图 3.3 所示。

从图中可见，Virtual Memory 那一项里的 MATLAB 整个程序运行的虚拟内存空间，由没有打开 3 GB 模式时候的 2 GB 变为打开 3 GB 模式后的 3 GB，增加了 1 GB。而在 Largest Contiguous Free Blocks 那一项里可看到两个在 1 000 MB 以上的连续地址空间。

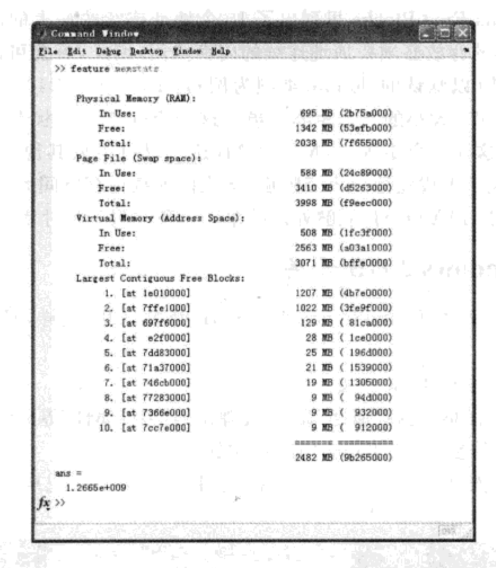


图 3.3 打开 3 GB 模式 MATLAB 运行内存环境参数

3.3 减小内存消耗注意事项

3.3.1 读取数据文件

当读取比较大的文件时，要注意选择那些必要的数据来读取。比如一个大的文本文件可能几万行、几十列，而我们需要的可能只有几列或者若干行。如果全部读取的话，虽然调用格式简单，但是非常占用内存，如果内存不够大的话，MATLAB 会提示“out of memory”错误。

读取文本文件时，如果格式复杂，并且文件体积较大的话，建议使用 textscan。textscan 是读取文本文档的底层函数，效率较高。举例来说，textscan 使用最多的调用格式如下：

```
data = textscan(fid, format, N, 'delimiter',',')
```

其中，fid 是 fopen(‘文件名’)后返回的文件句柄，用来标识文件中位置。format 是一字符串，形如%f %s%d…% * f% * s…% * [^\n]等。其中，%f 是按照浮点格式读相应的列的数据，%s 是按照字符串的格式读相应的列的数据，%d 是按照有符号整数读相应的列的数据，等等。% * 表示当前列的数据跳过不读，% * [^\n]表示从当前所在行的当前列开始到所在行的最后一列的数据都省略，并换行读取下一行数据；N 用来表示读取的行数；delimiter 后的字符表示区分列与列之间的标志。这样可以利用 textscan 函数读取想要的行和列的数据而忽略其他的。

此外，textscan 返回的数据存放在 cell 数组里，如果读取的全部是数值型的，可以利用 cell2mat 函数将其转化成数值矩阵以节省内存空间。

更详细的解释可以参看 textscan 的帮助文档。

此外，当读取大型二进制文件的时候，可以尝试使用 memmapfile 函数，这个函数的好处是利用虚拟内存技术，通过内存映射的方式读取二进制文件的数据，速度比 fread 和 fwrite 函数要快，非常适合经常要随机访问文件中各个位置数据的情形。

3.3.2 数据存储

这里说的数据存储指的是程序运行中各种变量在内存中的存储方式。应遵循以下原则：

(1) 根据问题需要选用适当的数据类型

在第 2 章已经比较了数值型矩阵和 cell 型以及 struct 型矩阵的存储特点和效率。其实数值型矩阵也有多种数据类型可供选择。譬如问题精度要求不高的话可以选择 single 类型来表示实数，一个 single 型的数据占用 4 B，比 double 型的少一半。MATLAB 中数值矩阵支持的数据类型见表 3.1。

表 3.1 MATLAB 数值类型表

类 型	说 明
int8	转换成带正负号、8 位的整数，其取值范围为[-128,127] 每个元素占用 1 B
int16	转换成带正负号、16 位的整数，其取值范围为[-32 768,32 767] 每个元素占用 2 B
int32	转换成带正负号、32 位的整数，其取值范围为[-2 147 483 648,2 147 483 647] 每个元素占用 4 B
int64	转换成带正负号、64 位的整数，其取值范围为[-9 223 372 036 854 775 808,9 223 372 036 854 775 807] 每个元素占用 8 B
uint8	转换成不带正负号、位的整数，其取值范围为[0,255] 每个元素占用 1 B
uint16	转换成不带正负号、16 位的整数，其取值范围为[0,65 535] 每个元素占用 2 B
uint32	转换成不带正负号、32 位的整数，其取值范围为[0, 4 294 967 295] 每个元素占用 4 B
uint64	转换成不带正负号、64 位的整数，其取值范围为[0, 18 446 744 073 709 551 615] 每个元素占用 8 B
single	转换成单精度浮点数 每个元素占用 4 B
double	转换成双精度浮点数 每个元素占用 8 B
char	转化成字符型 每个元素占 2 B

(2) 对于含零较多的矩阵采用稀疏矩阵来存储

sparse 函数可以用来产生稀疏矩阵。语句 S = sparse(i,j,s,m,n)能够利用向量 ***i***,***j***,***s*** 来产生一个 $m \times n$ 的稀疏矩阵 ***S***，具体的产生方法为 S(i(k), j(k)) = s(k)。其中 ***i***,***j*** 是 ***S*** 中不为 0 的元素的行索引和列索引，因此只能是正整数。此外，S = sparse(A)可以将矩阵 ***A*** 转化成对应的稀疏矩阵形式 ***S***。大家可以看下面的代码示例，体会一下稀疏矩阵在节省内存空间和提高计算速度方面的作用。

```
i = 1:1000;
j = randperm(1000); % 随机排列 1～1000 的所有整数
s = rand(1,1000); % 生成 1 * 1000 的服从(0,1)连续均匀分布的向量
S = sparse(i,j,s);
FS = full(S); % S 矩阵的非稀疏格式
whos
  Name        Size          Bytes  Class     Attributes

  FS       1000x1000      8000000  double
  S        1000x1000        16004  double     sparse
  i           1x1000         8000  double
  j           1x1000         8000  double
  s           1x1000         8000  double

>> tic;S2 = S * S;toc
Elapsed time is 0.000330 seconds.
>> tic;FS2 = FS * FS;toc
Elapsed time is 0.190742 seconds.
>> isequal(full(S2),FS2)
ans =
     1
```

从上面可以清楚看到，非稀疏矩阵 FS 需要 800 万个字节来存储，而其稀疏矩阵形式 S 仅需要 16 004 个字节来存储。计算速度方面，稀疏矩阵的计算速度远远高于非稀疏矩阵，稀疏化程度越高，这种差距越明显。此外，需要注意的是稀疏矩阵在求解一些大型的线性规划问题时用处很大，实际中很多大型的线性规划问题的约束矩阵的稀疏性都非常强，如果不采用稀疏矩阵形式来表示，惊人的计算量以及内存消耗量会使得在计算机上几乎无法求解大型规划问题。

最后有关稀疏矩阵使用要强调的一点是：应当在准备好数据后用 sparse 函数生成稀疏矩阵，而不要先创建一个大稀疏矩阵 S，然后在程序中用 S(i,j)的方式给它赋值。后者效率会很低。请看下面的比较。

```
clear;
i = 1:1000; % 行序号
j = randperm(1000); % 随机排列的列序号
n = rand(1,1000); % 相应位置的元素值
tic;S1 = sparse(i,j,n,1000,1000);toc % 用 sparse 函数生成稀疏矩阵时间
tic;
S2 = spalloc(1000,1000,1000);
for k = 1:1000
S2(i(k),j(k)) = n(k);
end;
toc % 先创建一个大稀疏矩阵 S,然后在程序中用 S(i,j)的方式给它赋值
Elapsed time is 0.000557 seconds.
Elapsed time is 0.010324 seconds.
```

由此可见后者效率比前者低得多。如果程序循环内有大量的这些操作，前后两种方法的运行时间差别是相当大的。

(3) 用 clear 清除掉不用的较大的变量

在程序运行过程中，往往会有一些变量不会再用到，但是这些变量依然在当前程序的工作

空间存在，如果这些变量较大并且较多，随着程序的运行就有可能发生内存溢出的错误提示。因此，需要在程序设计时对不会再用到的变量用 clear 清理掉。譬如下面的例子：

```
>> clear all
>> A = rand(1000);
sumA = sum(A);
whos
  Name        Size            Bytes  Class     Attributes

  A        1000x1000        8000000  double
  sumA        1x1000           8000  double

>> clear A
>> whos
  Name      Size            Bytes  Class     Attributes

  sumA      1x1000           8000  double
```

3.3.3 减小内存其他注意事项

MATLAB应用中还有其他一些技巧可以降低内存。例如：

(1) 反向量化操作

当操作的矩阵维数很大，矩阵占用的内存接近计算机的物理内存大小时，一些向量化的运算可能导致内存溢出错误。这时候就需要根据问题特点，采用循环的方式分批次将问题处理。

(2) 变量的共享问题

不同的函数空间，共享变量要比传递变量(无论值传递还是地址传递)效率要高，而且节省内存。MATLAB 7.0 之后可以利用嵌套函数(nested function)结构非常方便地实现不同函数间、主函数和子函数间的变量共享。关于嵌套函数的详细介绍，请参考第5章。

(3) 画图的问题

MATLAB在画图的时候都会将坐标数据存放在坐标轴句柄的'xdata'，'ydata'，'zdata'(三维的话)属性上，因此较多点的画图比较占用内存。在实际应用中，应该结合问题需要，来选取需要的数据进行画图。

(4) MATLAB 启动的问题

目前，MATLAB 的界面元素都是通过 Java 方式实现的。如果计算过程中对界面操作功能要求不高，纯计算占很大比重的话，可以通过运行 - nojvm 的方式启动 MATLAB，这样可以减少一百多兆的内存消耗。Windows 环境下具体启动方式如下：任务栏单击"开始"——"运行"按钮，输入"MATLAB - nojvm"，按 Enter 键即可。- nojvm 模式下的启动界面如图 3.4所示。

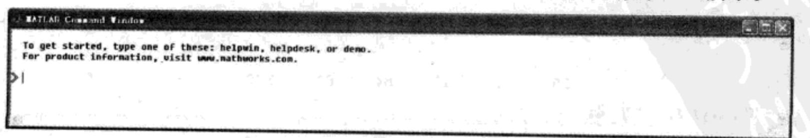


图 3.4 MATLAB - nojvm 启动模式界面

第 4 章

匿名函数类型

本章详细介绍 MATLAB 7.0 以后引入的新的函数类型之一——匿名函数(anonymous function)。匿名函数不但完全可以实现内联函数(inline function)的所有功能,而且一些用内联函数实现起来不大方便或者不简洁的功能,用匿名函数也非常容易实现。除此之外,通过 2.3.2 节的比较,可以清楚看到匿名函数的调用效率要远高于内联函数。

Mathworks 研发人员,包括首席科学家 Moler 教授本人都推荐用匿名函数来代替内联函数。Mathworks 之所以至今仍保留 inline 函数类型,主要就是照顾一些从 MATLAB 老版本时带过来的客户,使得他们开发的代码在新版本 MATLAB 上尽量具有较好的兼容性,同时也考虑到他们的使用习惯。但从效率角度以及易用性等方面来讲,无论是 MATLAB 初学者还是已使用 MATLAB 多年的人都应该掌握匿名函数的用法,并用它替代内联函数。

4.1 匿名函数

先来看下面几个问题:

1) 在程序编写过程中,经常需要计算诸如 $f(x)=\sin(x)/x$, $g(x,y)=\exp(x)+x^2\ln(y)$ 等这样一些数学表达式的值,如何方便地构造这些函数?写成 M 文件再调用肯定可以,但是从简洁以及管理上的方便性考虑这不是好办法。

2) 有时通过符号计算推导得到了一个函数表达式,接下来想通过数值计算对这个表达式进行进一步的操作,譬如求值,求数值积分等。如何进行?

3) 很多时候要对含参变量的函数进行操作,譬如 $f(x)=x^a$,如何在计算中得到 a 的具体值后再得到 $f(x)=x^a$ 的具体函数表达式?

上面这些问题都可以用匿名函数来方便的解决,事实上,匿名函数的应用非常灵活,读者在 2.3.2 节中已经初步体会了匿名函数的用法。下面将开始详细介绍。

4.1.1 匿名函数的基本定义

匿名函数的基本定义如下:

```
fhandle = @(arglist) expr
```

其中,expr 是具体的函数表达式,arglist 是指定的函数的自变量。具体请见下面示例:

```
f = @(x) x.^2;
>> fx = f(1:10)
fx =
     1     4     9    16    25    36    49    64    81   100
>> g = @(x,y) x.^2 + y.^2;
>> gxy = g(1:10,2:11)
gxy =
     5    13    25    41    61    85   113   145   181   221
```

4.1.2 匿名函数的种类

匿名函数按照不同的分类方法可以分为不同的种类。这里按照自变量的个数以及层数可以分为以下几种：单变量匿名函数、多变量匿名函数、单重匿名函数以及多重匿名函数。

1. 单变量匿名函数

单变量匿名函数是最简单的匿名函数，只含有一个自变量。例如 4.1.1 给的例子：

```
f = @(x) x.^2;
```

就是单变量匿名函数，除此之外，含有参数，参数值已知的单个自变量的匿名函数也是单变量匿名函数，譬如：

```
a = 10;
b = 20;
f = @(x) a * x + b;
>> f(1:5)
ans =
    30    40    50    60    70
```

2. 多变量匿名函数

多变量匿名函数含有两个或两个以上的自变量，例如 4.1.1 给的例子：

```
g = @(x,y) x.^2 + y.^2;
```

同单变量匿名函数一样，多变量匿名函数也可以有已知的参数值。譬如：

```
>> a = 1;
b = 2;
g = @(x,y) a * x + y.^b;
g(1:5,1:5)
ans =
    2    6    12    20    30
>>
```

3. 单重匿名函数

到目前为止，上面列的匿名函数，无论单变量还是多变量匿名函数，都属于单重匿名函数。这类匿名函数的特点是：只有一个“@”符号引导，“@”符号之后就是具体的函数表达式。自变量输入单重匿名函数后，得到的是具体的数值。除了单重匿名函数外，还有二重、乃至多重匿名函数。这些多重匿名函数在参数传递方面非常方便。

4. 多重匿名函数

下面以二重匿名函数为例来介绍多重匿名函数，读者可以参考二重匿名函数写出多重匿名函数。下例是一个简单的二重匿名函数：

```
f = @(a,b) @(x) a * x + b
f =
    @(a,b)@(x)a * x + b
```

其中“a,b”是外层变量，“x”是内层变量。可以这样理解这个表达式：每个“@”符号后面括号里的变量的作用域一直到表达式的结尾。这样，“a,b”的作用域就是“@(x) a * x+b”，而“x”的作用域就是“a * x+b”。因此，对于给定的“a,b”，“gab=f(a,b)”是一个单层以 x 为变量的单变量匿名函数。

可以利用 functions 函数观察建立的匿名函数信息，如下：

```
f = @(a,b) @(x) a * x + b
f =
    @(a,b)@(x)a * x + b
>> f23 = f(2,3)
f23 =
    @(x)a * x + b
>> f23info = functions(f23)
f23info =
    function: '@(x)a * x + b'
        type: 'anonymous'
        file: ''
    workspace: {3x1 cell}
>> f23info.workspace{1}
ans =
1x1 struct array with no fields.
>> f23info.workspace{2}
ans =
    varargout: {[1x1 function_handle]}
           a: 2
           b: 3
>> f23info.workspace{3}
ans =
1x1 struct array with no fields.
```

两重以上的匿名函数可以参考二重匿名函数类推，譬如：

```
>> f = @(a) @(b,c) @(x) x^a + b * c
f =
    @(a)@(b,c)@(x)x^a + b * c
```

两重以上的匿名函数各变量的作用域可以参考二重匿名函数。多重匿名函数在变量传递方面非常方便，本章后半部分将给出具体应用的例子。

4.2 匿名函数应用实例

本节将通过一系列例子来讲述匿名函数的用法。这些例子很多都是源自一些 MATLAB 技术论坛网友提出的问题，很多问题具有普遍性，都是比较典型的，相信读者可以从中得到启发。

4.2.1 匿名函数在求解方程中的应用

匿名函数可以非常方便地表示所求方程，并供 fzero 等求解函数调用。请看下面的例子。

【例 4.2-1】 求下列方程的根：

$$f(x)=\mathrm{e}^x+x^2+x^{\sqrt{x}}=100$$

求解代码如下：

```
>> f = @(x) exp(x) + x^2 + x^(sqrt(x)) - 100        % 构造方程的匿名函数形式
f =
     @(x)exp(x) + x^2 + x^(sqrt(x)) - 100
>> format long
>> x0 = fzero(f,3)                                  % 求方程的根,初始值为 3
x0 =
     4.163549956946139                              % 求出的解
>> f(x0)                                            % 代入原方程验证
ans =
     2.842170943040401e - 014
```

从例 4.2-1 中可以看出，匿名函数在表达方程方面很简洁。例中没有参数，如果方程有额外的参数，而且要求对不同的参数一一求解方程相应的根，该怎么办呢？请看例 4.2-1 续。

【例 4.2-1 续】 对于 $a=[0,0.01,0.02,\cdots,2]$，求下列方程相应的 x 值，并画出 a 和相应的 x 的图像。

$$f(x)=\mathrm{e}^x+x^a+x^{\sqrt{x}}=100$$

本例多了一个参数项 a，因此对于不同的 a，方程的具体表达式也不一样。下面的代码给出这一求解并且画图的过程。

```
f = @(a) @(x) exp(x) + x^a + x^(sqrt(x)) - 100; % 构造函数句柄
format long
aa = 0:0.01:2;
plot(aa,arrayfun(@(a) fzero(f(a),4),aa),'* -') % 利用 arrayfun 求解不同的 a 对应的 x
xlabel('$ a $','interpreter','latex','fontsize',15) % 标注 x,y 坐标轴,按照 latex 语法
ylabel('$ x $','interpreter','latex','fontsize',15)
title('$ \mathrm{e}^{x} + x^{\sqrt{x}} + x^a - 100 $','interpreter','latex',...
'fontsize',15)
```

生成的图形如图 4.1 所示。

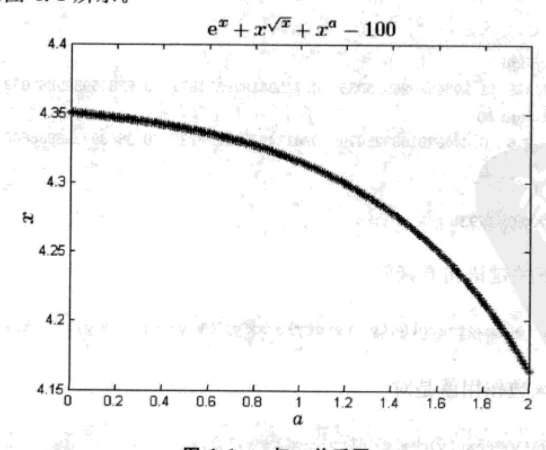


图 4.1 a 与 x 关系图

4.2.2 匿名函数在显式表示隐函数方面的应用

隐函数一般无法在数学上显式表示。这里说的显式表示指的是构造一个 MATLAB 函数来表达隐函数，具体思路是对于给定的隐函数的自变量 x，通过数值方法求解出因变量 y，这样就相当于显式表达隐函数。请看下面的例子：

【例 4.2－2】 显式表示下列 y 关于 x 的隐函数：

$$(e^y+x^y)^{\frac{1}{y}}-x^2y=0$$

利用匿名函数，可以在 MATLAB 中显式得写出 y 和 x 的关系式如下：

```
y = @(x) fzero(@(y) (exp(y) + x^y)^(1/y) - x^2 * y,1);
```

这样，对于任意的 x，只需调用 $yx=y(x)$，就能得到对应的 y 值，如：

```
format long
y1 = y(1)
y1 =
    2.777942350124938
>> y2 = y(2)
y2 =
    1.105452026515033
>> y3 = y(3)
y3 =
    0.775941879211877
```

这时的 y 只能接受标量 x 输入，利用函数 arrayfun，也可以令其接受向量输入：

```
>> format long
>> Y = @(x) arrayfun(@(xx)fzero(@(y) (exp(y) + xx^y)^(1/y) - xx^2 * y,1 ), x);
>> Y(1:10)
ans =
    Columns 1 through 5
    2.777942350124938  1.105452026515033  0.775941879211877  0.628359329251039  0.542541817671730
    Columns 6 through 10
    0.485590680913226  0.444620168925079  0.413493851208235  0.388897398866600  0.368874717117660
    >> Y(1)
    ans =
        2.777942350124938
```

读者可能对如下关键语句有疑问：

```
Y = @(x) arrayfun(@(xx)fzero(@(y) (exp(y) + xx^y)^(1/y) - xx^2 * y,1 ), x);
```

其实 arrayfun 函数作用就是对

```
y = @(x) fzero(@(y) (exp(y) + x^y)^(1/y) - x^2 * y,1 );
```

增加了一个循环的“外壳”，使得 Y 既可以接受标量 x 输入，还可以接受向量 x 的输入。其中，

```
@(xx)fzero(@(y) (exp(y) + xx^y)^(1/y) - xx^2 * y,1 )
```

是 arrayfun 函数的第一个输入参数，定义了以循环变量“xx”为输入参数的匿名函数，实现的功能是针对每个 xx，求对应的使

$$(e^y+(xx)^y)^{\frac{1}{y}}-(xx)^2y=0$$

成立的 y。xx 的循环范围是 Y 的输入变量 x，也就是上述 arrayfun 函数的第二个输入参数，这样 x 是标量的时候 xx 的循环范围就是一个值，是向量的时候就遍历 x 的每个元素。

【例 4.2-3】 显式表示下列 z 关于 x,y 的隐函数：

$$z=\sin\left((zx-0.5)^2+2xy^2-\frac{z}{10}\right)\exp\left(-\left((x-0.5-\exp(-y+z))^2+y^2-\frac{z}{5}+3\right)\right)$$

借助匿名函数，可以写出 z 关于 x,y 的关系式如下：

```
z = @(x,y) fzero(@(z) z - sin((z * x - 0.5)^2 + x * 2 * y^2 - z/10) * ...
exp( - ((x - 0.5 - exp( - y + z))^2 + y^2 - z/5 + 3)),rand);
```

其中，fzero 函数求解该隐函数的初值为随机值 rand。利用上述匿名函数，可以画出 z 关于 x，y 的图形：

```
[X,Y] = meshgrid( - 1;0.1;7, - 2;0.1;2);
Z = arrayfun(@(x,y) z(x,y),X,Y);
surf(X,Y,Z)
xlabel('\fontsize{15}\fontname{times new roman}x','color','b')
ylabel('\fontsize{15}\fontname{times new roman}y','color','b')
zlabel('\fontsize{15}\fontname{times new roman}z','color','b')
title('\fontsize{15}\fontname{隶书}z(x,y)的函数图像','color','r')
```

画出的图形如图 4.2 所示。

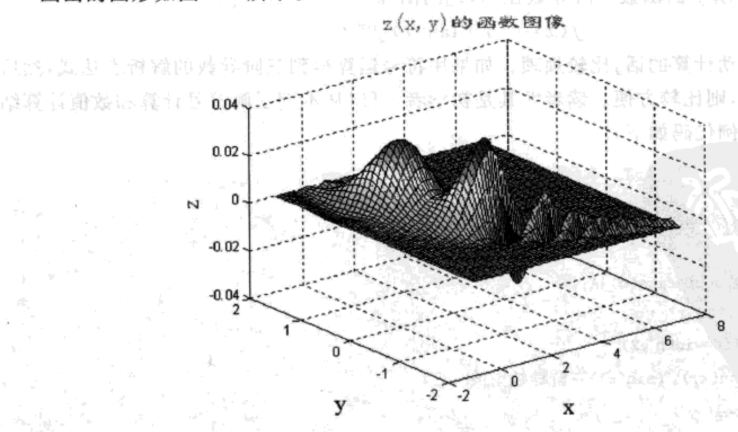


图 4.2 $z(x,y)$ 关于 x,y 的图像

4.2.3 匿名函数在求积分区域方面的应用

有的时候，需要根据已知的积分值和被积函数求对应的积分区域，当被积函数表达式不是很复杂的时候，用匿名函数比较适合。请看下面例题：

【例 4.2-4】 要使 $\sin^2(x)/x^2$ 这个式子的积分值为 0.99π，求其关于 0 对称的积分区域。

该积分区域相对 y 轴是对称的，故实际计算时，只需计算 $x>0$ 范围的积分区域。本例利用匿名函数求解的话非常方便，仅有一行代码：

```
u0 = fzero(@(u) 0.99 * pi/2 - quadl(@(x) sin(x).^2./(x.^2),0,u), 1)
u0 =
   32.3138
```

上面代码的意思是求"0.99 * pi/2－quadl(@(x) sin(x).^2./(x.^2),0,u)"等于 0 成立的 u，fzero 求解的时候设置的初始值为 1，最后求出的结果是 u0。

4.2.4 匿名函数在求数值积分方面的应用

匿名函数在求数值积分的方面应用非常灵活，功能也非常强大。本书第 6 章将专门讨论各种类型的积分以及积分方程求法，那里会有大量匿名函数应用的例子。

4.2.5 匿名函数和符号计算的结合

有的时候需要推导一些表达式，而对得到的表达式进行诸如求值、积分或者极值的运算。如果表达式不是很复杂的话，可以手动推导。但是很多时候手动推导非常烦琐甚至根本不可行，这时候就要借助计算机来推导。在得到表达式后，再将其转化成匿名函数，从而方便的进行后续处理。请看下面的例子：

【例 4.2-5】 求下面函数三阶导数在[0,1]的图像。

$$f(x)=[x+\tan(x)]^{\sin(x)}$$

本例如果用手动计算的话，比较烦琐。如果用符号运算得到三阶导数的解析表达式，然后再转化成匿名函数，则比较方便。读者尤其是初学者，可以从本例了解符号计算和数值计算结合的一种途径。本例代码如下：

```
syms x
f = (x + tan(x))^(sin(x));
c = diff(f,3);
f3 = eval(['@(x)' vectorize(c)]);
x = linspace(0,1,100);
plot(x,f3(x),'linewidth',2)
title('y = (x + tan(x))\^(sin(x))三阶导数图像')
xlabel('x');ylabel('y')
```

最后得到的图形如图 4.3 所示。

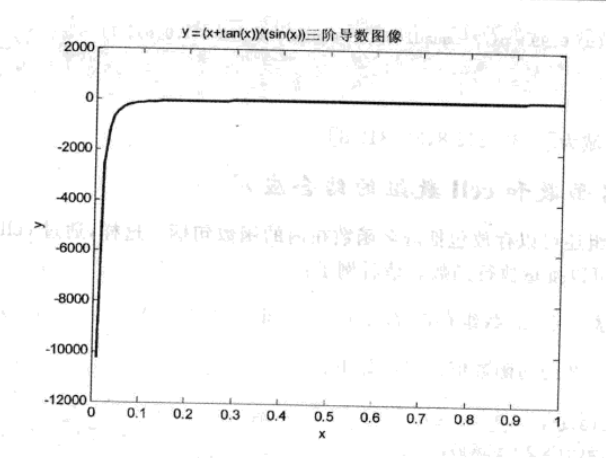


图 4.3　$[x+\tan(x)]^{\sin(x)}$三阶导数图像

4.2.6　匿名函数在优化中的应用

匿名函数在优化中的应用主要是以表示目标函数的形式出现的。譬如下例：

【例 4.2-6】　求下面函数的最小值：

$$f(x)=3x_1^2+2x_1x_2+x_2^2$$

用匿名函数来表示目标函数 $f(x)$ 如下：

```
f = @(x) 3 * x(1)^2 + 2 * x(1) * x(2)  + x(2)^2;
```

进一步求解：

```
x0 = [1,1]; % 初始值
[x,fval] = fminunc(f,x0)
x =
   1.0e-006 *
     0.2541   -0.2029
fval =
   1.3173e-013
```

在默认精度下，得到最优解为 $x_1=2.541\times10^{-7}$，$x_2=-2.029\times10^{-7}$，最优值为 $fval=1.3173\times10^{-13}$。

4.2.7　匿名函数在求积分区域方面的应用

有的时候，需要根据特定的积分值反求积分区域。例如下面的问题：

【例 4.2-7】　使 $(\sin^2 x)/x^2$ 这个式子的积分值为 0.99π，求在 MATLAB 上实现求解其积分域。

该积分区域相对 y 轴是对称的，故实际计算时，只需计算 $x>0$ 范围的积分区域。求解代码如下：

```
u0 = fzero(@(u) 0.99 * pi/2 - quadl(@(x) sin(x).^2./(x.^2),0,u), 1)
u0 =
    32.3138
```

得到积分区域为[－32.313 8，32.313 8]

4.2.8 匿名函数和 cell 数组的结合应用

cell 型的数组还可以存放包括匿名函数在内的函数句柄。这样，通过 cell 数组，可以存储多个函数，并且可以批量执行函数。请看例子：

【例 4.2－8】 用 cell 数组存储 $f_1(x)=x^2+\sin(x)$，$f_2(x)=x^3-5$，$f_3(x)=\sin\left(\frac{1}{\sqrt{x}}\right)$，并求各个函数在 $x=2$ 处的函数值。代码如下：

```
>> fun = cell(3,1);
>> fun{1} = @(x) x^2 + sin(x);
>> fun{2} = @(x) x^3 - 5;
>> fun{3} = @(x) sin(1/sqrt(x));
>> val = cellfun(@(x) x(2),fun)
val =
    4.9093
    3.0000
    0.6496
```

读者请注意 cellfun 函数，可以看到，cellfun 对每个 fun 中存储的函数都进行了求其在 $x=2$处的值的操作。得到的值分别为 $f_1(2)=4.909\,3$，$f_2(2)=3.000\,0$，$f_3(2)=0.649\,6$。

第 5 章

嵌套函数类型

本章详细介绍 MATLAB 7.0 以后引入的另一个强大的函数类型——嵌套函数类型(nested function)。嵌套函数和子函数既有区别也有联系，嵌套函数可以方便地实现变量共享。这样在进行参数传递时，使得程序更加灵活。

第 4 章详细地介绍了匿名函数的用法，读者对匿名函数的方便易用已经有了一些体会，而嵌套函数更加灵活和强大，可以实现匿名函数实现起来不能或者很烦琐的功能。

5.1 嵌套函数

类似第 4 章，先来看下面几个问题：

1) 在程序编写过程中，有的数学表达式比较长，不方便用一行来表示，而且这个表达式里还有其他的参数。这时候就需要将其写成函数，用嵌套函数可以方便的解决参数共享问题。

2) 在求解微分方程或者进行优化问题求解时，一些复杂的表达式往往涉及参数传递过程，这个时候，比较适合将其表达式写成嵌套函数。

3) 在编写 GUI(Graphical User Interface 的缩写，即：图形用户界面，简称 GUI。)的回调函数时，参数传递往往是个棘手的问题。利用嵌套函数，可以方便解决这个问题。

嵌套函数结构非常强大，方便易用。读者在 2.3.2 节中已经初步体会了嵌套函数的用法。下面将开始详细介绍。

5.1.1 嵌套函数的基本定义

嵌套函数，即 nested function，顾名思义，是嵌套在函数体内部的函数。嵌套函数可以出现在一个函数体内部的任何位置，开始以 function 声明，结束的时候加上 end。需要说明的是，包含有嵌套函数的函数，无论它是主函数、子函数或者嵌套函数，都应该在末尾的地方加上 end。下面代码是一个简单的嵌套函数的例子：

```
function r = MyTestNestedFun(input)
a = 5;
c = sin(input) + tan(input);
function y = nestedfun(b)
y = a * c + b;
end
r = nestedfun(5);
end
```

保存后在命令窗口中输入：

```
r = MyTestNestedFun(6)
```

得到如下结果：

```
r =
   2.1479
```

结果是这样得到的：调用 MyTestNestedFun 后，程序依次执行 a=5；c=sin(input)+tan(input)；然后又调用 nestedfun 这个嵌套函数，此时 b=5，而嵌套函数所在的函数中的 a，c 对嵌套函数是可见的。即 a=5，c=−0.5704。于是 r=a*c+b=5*(−0.5704)+5=2.1479。

5.1.2 嵌套函数种类

嵌套函数可以分为单重嵌套函数和多重嵌套函数。先说单重嵌套函数，这样的函数嵌套在别的函数体内，自己内部不再有嵌套的函数，如 5.1.1 节中给出的函数示例。一个函数里可以有一个或者多个单重嵌套函数。

多重嵌套函数，这样的函数嵌套在别的函数体内，同时自己内部又嵌套着别的另一层或几层函数。同样一个函数里可以有一个或者多个多重嵌套函数。如下面类型的函数：

```
function x = A(p1, p2)
...
    function y1 = B1(p3)
    ...
        function z1 = C1(p4)
        ...
        end
    ...
    end
...
    function y2 = B2(p5)
    ...
        function z2 = C2(p6)
        ...
            function w = D(p7)
            ...
            end
        end
    end
end
```

该函数 A(p1，p2)内部有两个嵌套函数，分别是 B1(p3)和 B2(p5)，而 B1(p3)和 B2(p5)分别是二重和三重嵌套函数。

5.2 嵌套函数的变量作用域

变量的作用域指的是变量能够被程序访问、修改、设置等的代码范围。含有嵌套函数的函

数体内的变量作用域非常重要但同时又容易搞错。在讨论这些变量作用域前，请读者先运行下面三个例子，看一下结果，体会一下位于不同位置的变量的作用域。

【例 5.2-1】 变量作用域示例 1。

```
function r = NestedFunctionVarScopeDemo(a)
b = a + 1;
    function Nested1
        c = b + 1;
        function Nested11
            d = c + a;
        end
        Nested11;
    end
Nested1
r = d;
end
```

运行上面的代码，譬如：

```
r = NestedFunctionVarScopeDemo(1)
r =
    4
```

上述代码有一个二重嵌套函数 Nested1，它内部还包含一个嵌套函数 Nested11，整个函数的执行过程如下：

传入变量 a=1，计算 b 的值，b=2；遇到 Nested1 的函数定义，往下是调用 Nested1 的语句，进入 Nested1 之后先计算 c 的值，即 c 等于 b+1=3。从这里可以看到，在嵌套函数体内，可以访问主函数之内的变量。"c=b+1;"这条语句之后是定义 Nested11 的函数语句，定义 Nested11 结束后，是调用 Nested11 的函数语句。从 Nested11 的定义来看，这个函数非常简单，仅仅做的是计算 d=c+a；从这里可以看出在第二重嵌套函数里，分别访问了 Nested1 里的变量 c 和主函数里的变量 a。主函数最后一句是从外部访问了第二重嵌套函数里的变量 d，并将 d 赋给 r，以此作为主函数的返回值。

从上面这个嵌套函数的示例可以看出，主函数和嵌套在其内的函数，它们各自的变量是可以互相访问的。但是必须注意的是，嵌套函数访问主函数的变量，可以在函数定义里直接拿过来用，而主函数访问嵌套在其内的函数里的变量则必须要经过调用之后才能用。譬如上例主函数访问 Nested11 里的 d，是经过在主函数里调用 Nested1，而在 Nested1 里又调用 Nested11 后才成功的，否则是不行的。再看下面的例子。

【例 5.2-2】 变量作用域示例 2。

```
function r = NestedFunctionVarScopeDemo2(a)
b = a + 1;
    function Nested1
        c = b + 1;
        c1 = 10;
        function Nested11
```

```
            d = c + a;
        end
    end
Nested1
e = c1
r = d;
end
```

运行得到如下结果：

```
r = NestedFunctionVarScopeDemo2(1)
e =
    10
??? Undefined function or variable "d".

Error in ==> NestedFunctionVarScopeDemo2 at 12
r = d;
```

在调用 NestedFunctionVarScopeDemo2 时候发生了错误，从提示来看，是没有定义变量“d”。这是因为调用 Nested1 后执行了 Nested1 里的语句，但是 Nested1 中只是定义了 Nested11，而并没有调用它的语句。因此，外界不能访问 d。这也就是为什么 e＝c1；可以成功而 r＝d；报错的原因了。

上面讨论了嵌套函数和主函数之间变量互相访问的情况（包括主函数也是嵌套函数的情况）。那么不同的嵌套函数之间呢，它们之间没有嵌套关系变量能不能互相访问呢？请看下面的例子。

【例 5.2-3】 变量作用域示例 3。

```
function r = NestedFunctionVarScopeDemo3(a)
b = a + 1;
    function Nested1
        c = b + 1;
        c1 = 10;
        Nested2;
        c2 = d^2;
    end
    function Nested2
        d = 2 * b;
    end
Nested1
r = c2
end
```

该函数里面包含两个嵌套函数，都是单重的，一个是 Nested1，一个是 Nested2。本例是想验证，能否直接从 Nested1 中通过调用 Nested2 来访问其中的变量。经过运行得到下面的结果：

```
r = NestedFunctionVarScopeDemo3(1)
??? Undefined function or variable 'd'.

Error in ==> NestedFunctionVarScopeDemo3 > Nested1 at 7
        c2 = d^2;

Error in ==> NestedFunctionVarScopeDemo3 at 12
Nested1
```

从错误提示来看，出错原因是访问变量“d”不成功，“d”未定义，所以彼此没有嵌套关系的嵌套函数间是不能简单得共享变量的。如果非要共享，只能通过它们所在的主函数来进行。

5.3 嵌套函数彼此调用关系

5.2节讨论了嵌套函数的变量作用域。在讨论过程中读者可能已经注意到了，主函数和嵌套函数之间的变量共享需要通过调用来实现。那么这就涉及一个问题，主函数和嵌套函数，不同的嵌套函数之间，以及不同嵌套层次的函数，它们之间的调用关系是怎样的呢？本节就来详细讨论这些问题。

5.3.1 主函数和嵌套函数之间

这里的主函数包括最上层的主函数，即不嵌套在任何函数里的函数以及本身是嵌套函数，但同时内部还嵌套有其他函数的函数。因为这些函数相对嵌套在内的函数来说也是主函数。这样的情形下，调用关系遵循下面的原则：主函数可以调用嵌套在其中的第一重嵌套函数，而不能调用第二重或者更深重的嵌套函数；无论第几重嵌套函数，都可以调用其主函数或者主函数的主函数等等。请看下面的例子：

【例 5.3-1】 嵌套函数调用示例1。

```
function r = NestedFunctionCallDemo1(a)
b = a + 1;
    function c1 = Nested1(x)
        c = b + 1;
        c1 = 10 + c * x;
        function d = Nested11
            d = c + a;
        end
    end
c1 = Nested1(1),
r = Nested11;
end
```

本例呈现的是主函数调用子函数的例子，从例5.3-1可以看出，在主函数 NestedFunctionCallDemo1 里分别对 Nested1 和 Nested11 进行了调用。运行结果如下：

```
r = NestedFunctionCallDemo1(1)
c1 =
    13
??? Undefined function or variable 'Nested11'.

Error in ==> NestedFunctionCalleDemo1 at 11
r = Nested11;
```

从结果来看，调用 Nested1 成功，而调用 Nested11 没有成功。这验证了"主函数可以调用嵌套在其中的第一重嵌套函数，而不能调用第二重或者更深重的嵌套函数"。

如果是嵌套函数调用主函数的情形呢？再看下面这个例子：

【例 5.3-2】 嵌套函数调用示例 2。

```
function  NestedFunctionCallDemo2(flag)
switch flag
    case 1
        disp('flag = 1');
        return;
    case 2
        disp('flag = 2');
        NestedFun1
    case 3
        disp('flag = 3');
        return
    otherwise
        disp(['flag = ',num2str(flag)]);
        return
end
    function NestedFun1
        NestedFunctionCallDemo2(1);
        NestedFun2
        function NestedFun2
            NestedFunctionCallDemo2(3)
        end
    end
end
```

运行上述代码，得到下面结果：

```
NestedFunctionCallDemo2(2)
flag = 2
flag = 1
flag = 3
```

当 flag=2 时，执行的是 NestedFun1，而在 NestedFun1 中调用了主函数，这时候 flag=1，因此显示"flag=1"，接下来在 NestedFun1 中对 NestedFun2 进行了调用。而 NestedFun2 对最外层的主函数来说是一个二重嵌套函数，同样它也调用了 NestedFunctionCallDemo2，只不过 flag=3，因此显示"flag=3"。从例 5.3-2 可以看出，嵌套函数对主函数的调用是可以的。

5.3.2 不同的嵌套函数之间

这里讨论的不同嵌套函数之间指的是嵌套深度相同或者不同，彼此之间没有嵌套与被嵌套关系的不同的嵌套函数。譬如位于第一层的不同嵌套函数之间，以及位于第二层的不同嵌套函数之间，等等。请看下面这个例子：

【例5.3-3】 嵌套函数调用示例3。

```
function  NestedFunctionCallDemo3
Nested1(5)
    function Nested1(x)
        disp(['Nested1 执行,输入:',num2str(x)])
        Nested2(6)
        function Nested11(xx)
            disp(['Nested11 执行,输入:',num2str(xx)]);
        end
    end
    function Nested2(y)
        disp(['Nested2 执行,输入:',num2str(y)])
        function Nested22(yy)
            disp(['Nested22 执行,输入:',num2str(yy)]);
        end
    end
end
```

运行结果如下：

```
Nested1 执行,输入:5
Nested2 执行,输入:6
```

从上面结果来看，Nested1 执行后成功调用 Nested2，这说明第一层的嵌套函数之间是可以互相调用的。那么第二层的嵌套函数之间以及第二层嵌套函数和第一层嵌套函数之间呢？

【例5.3-4】 嵌套函数调用示例4。

```
function NestedFunctionCallDemo4
Nested1(5)
    function Nested1(x)
        disp(['Nested1 执行,输入:',num2str(x)])
        Nested11(6)
        function Nested11(xx)
            disp(['Nested11 执行,输入:',num2str(xx)]);
            Nested2(pi)
            Nested22(10);
        end
    end
    function Nested2(y)
        disp(['Nested2 执行,输入:',num2str(y)])
        Nested22(pi * pi)
        function Nested22(yy)
```

```
            disp(['Nested22 执行,输入:',num2str(yy)]);
        end
    end
end
```

运行代码，得到如下结果：

```
Nested1 执行,输入:5
Nested11 执行,输入:6
Nested2 执行,输入:3.1416
Nested22 执行,输入:9.8696
??? Undefined function or method 'Nested22' for input arguments of type 'double'.

Error in ==> NestedFunctionCallDemo4 > Nested1/Nested11 at 9
            Nested22(10);

Error in ==> NestedFunctionCallDemo4 > Nested1 at 5
        Nested11(6)

Error in ==> NestedFunctionCallDemo4 at 2
Nested1(5)
```

从上面的执行情况可以看出，函数 Nested11 成功被调用，而且 Nested11 再调用 Nested2 时也成功了，通过 Nested2，间接调用了 Nested22，但是从 Nested11 中直接调用 Nested22 却没有成功。这说明第二重嵌套函数可以调用不包含它的第一重嵌套函数。如果是第三重嵌套函数，那么能否调用不包含它的第二重嵌套函数呢？请看下面例子：

【例 5.3-5】 嵌套函数调用示例 5。

```
function NestedFunctionCallDemo5
Nested1(5)
    function Nested1(x)
        disp(['Nested1 执行,输入:',num2str(x)])
        Nested11(6)
        function Nested11(xx)
            disp(['Nested11 执行,输入:',num2str(xx)]);
            Nested111(pi)
            function Nested111(xxx)
                disp(['Nested111 执行,输入:',num2str(xxx)]);
                Nested2(exp(1))
                Nested22(100)
            end
        end
    end
    function Nested2(y)
        disp(['Nested2 执行,输入:',num2str(y)])
        Nested22(pi * pi)
        function Nested22(yy)
            disp(['Nested22 执行,输入:',num2str(yy)]);
        end
    end
end
```

运行【例 5.3-5】,得到如下结果:

```
Nested1 执行,输入:5
Nested11 执行,输入:6
Nested111 执行,输入:3.1416
Nested2 执行,输入:2.7183
Nested22 执行,输入:9.8696
??? Undefined function or method 'Nested22' for input arguments of type 'double'.

Error in ==> NestedFunctionCallDemo5 > Nested1/Nested11/Nested111 at 12
        Nested22(100)

Error in ==>NestedFunctionCallDemo5 > Nested1/Nested11 at 8
        Nested111(pi)

Error in ==> NestedFunctionCallDemo5 > Nested1 at 5
        Nested11(6)

Error in ==> NestedFunctionCallDemo5 at 2
Nested1(5)
```

从上面的执行情况可以看出,函数 Nested1 执行后,函数 Nested11 成功被调用,而且 Nested11 再调用 Nested111,Nested111 再调用 Nested2 时也成功了;通过 Nested2,Nested111 间接调用了 Nested22,但是从 Nested111 中直接调用 Nested22 却没有成功。这说明第三重嵌套函数,不可以调用不包含它的第二重嵌套函数。

5.3.3 嵌套函数调用关系总结

本节前半部分讨论了各种情形下的嵌套函数相关调用情况,为了方便读者理解并记忆这些调用关系,可以将上述调用情况进行类比。

具体如下:将主函数看成"父亲",嵌套函数依据嵌套深度可以看成"儿子","孙子","重孙",等等。这样最外层的主函数可以认为是所有内部函数的"祖先",位于不同嵌套函数里的函数之间的关系类似于一个家族之间的"亲属关系"。

譬如例 5.3-5 中,"祖先"是函数 NestedFunctionCallDemo5,函数 Nested1 和函数 Nested2 是它的两个"儿子",函数 Nested11 和函数 Nested22 是它的两个"孙子",而函数 Nested111 是它的重孙子。函数 Nested11 和 Nested11 之间是"堂兄弟"的关系等等。把"函数调用另一个函数"类比于"一个人求助于另一个人"。这样嵌套函数之间的调用关系可以总结成下面几句话:

父亲可以求助儿子,儿子可以求助父亲,也即父子可以互相求助。一个人不能求助孙子、重孙等后代,但是可以求助自己的祖宗(祖父、曾祖父等等)。一个人可以求助自己的亲兄弟,或者叔叔、伯伯但不能求助侄儿。

5.4 嵌套函数应用实例

本节将通过一系列例子来展示嵌套函数的用法，通过这些例子，读者可以体会嵌套函数的方便以及强大。这些例子很多是在各MATLAB技术论坛被网友频繁提问的一些问题。相信读者可以从中获得启发。

5.4.1 嵌套函数在求解积分上限中的应用

【例5.4-1】 如下述积分表达式，已知 a、e 和 l，如何求得 β_0？

$$l=\int_0^{\beta_0}\frac{a(1-\mathrm{e}^2)}{(1-\mathrm{e}^2\sin^2\beta)^{3/2}}\mathrm{d}\beta$$

本例关于 β 的积分结果不能解析表示，需要用数值积分来做，同时还要求解一个非线性方程，本例积分表达式写出来不是很复杂，可以利用匿名函数来做。有兴趣的读者可以尝试下，这里采用嵌套函数来求解。代码如下：

```
function sol = example541(a,e,l)
    function f = fun1(beta)
            f = a. * (1 - e.^2)./(1 - e.^2. * sin(beta).^2).^(3/2);
        end
    function g = fun2(beta0)
        g = quadl(@fun1,0,beta0) - l;
    end
sol = fzero(@fun2,3);
end
```

本例采用了两个嵌套函数，fun1 和 fun2，fun1 的功能是建立积分表达式，fun2 的功能是建立关于 β_0 的非线性方程，由于 a、e 和 l 是已知的，因此，a、e 和 l 作为整个函数 example541 的输入，实际应用中给定一组 a、e 和 l 的值就可以求出对应的 beta0 的值。譬如给定 $a=20$、$e=0.6$ 和 $l=6$，可以求解相应的解如下：

```
sol = example541(20,0.6,6)
sol =
    0.4519
```

5.4.2 嵌套函数在GUI中的应用

利用嵌套函数可以方便共享变量的特点，在编写GUI中的回调函数(callback function)时，可以采用嵌套函数来实现。下面举一个嵌套函数在GUI中应用的例子：

【例5.4-2】 用MATLAB生成一个如下的三角形界面，如图5.1所示。

要求：

1) 图上的数字根据行数和列数动态生成，每一层都比上面大1。

2) 在滚动条拖动的时候，会根据滚动条的位置，使三角形中的数字变红，最大值使数字全部变为红色，最小值使得数字全变为部黑色。

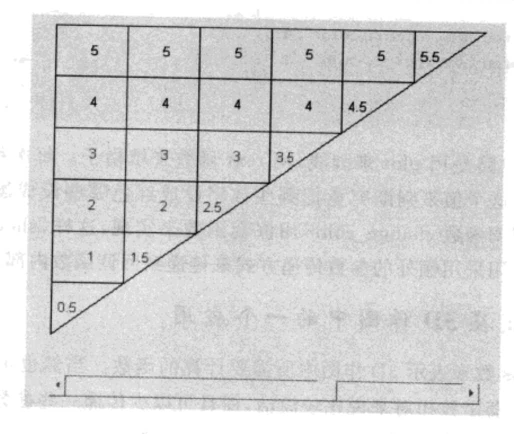


图 5.1 三角界面示意图

本例的代码如下：

```
function triangle_table
fig = figure('defaultuicontrolunits','normalized','name','triangle_table',...
    'numbertitle','off','menubar','none');
ah = axes('Pos',[.1 .2 .75 .75],'Visible','off');
slider_h = uicontrol('style','slider','units','normalized','pos',...
[0.1,0.05,0.75,0.05],'sliderstep',[1/6,0.05],'callback',@change_color);
hold on
for k = 0:6
    plot(0:6 - k,(6 - k) * ones(1,(7 - k)),'k');
    plot(k * ones(1,(7 - k)),k:6,'k');
end
plot([0,6],[0,6],'k');
hold off;
for x = 1:5
    for y = 1:x
        text(y - 0.5,x + 0.5,num2str(x),'color','k','tag','数字');
    end
end
for k = 0:5
    text(k + 0.1,k + 0.5,[num2str(k),'.5'],'tag','数字');
end
% ==== slider's callback function(nested function) ======
    function change_color(hObject,eventdata)
        v = round(6 * get(slider_h,'value'));
        num_h = findobj('tag','数字');
        num_pos = get(num_h,'pos');
        red_num_logic = cellfun(@(x) (x(1)<= v&&x(2)<= v),num_pos);
```

```
        set(num_h(red_num_logic),'color','r');
      set(num_h(~red_num_logic),'color','k');
    end
end
```

本例生成界面的思路是用 plot 来画线，用 text 函数来填格子。每次移动滚动条后，得到当前滚动条的值，根据这个值来判断究竟把哪些数字设成红色哪些设成黑色的。slider_h 是滚动条的句柄，它的回调函数 change_color 用嵌套函数来实现，这样，slider_h 对于嵌套函数内部来说是可见的，不用采用额外的参数传递方式来传递到回调函数内部。

5.4.3 嵌套函数在 3D 作图中的一个应用

本例是利用嵌套函数来表示 3D 作图中所需要计算的函数。当然也可以采用其他函数形式来实现，但是采用嵌套函数相对来说比较简洁，而且可以少传递一些参数。例子如下：

【例 5.4－3】 画出下列函数的图像：

$$T(n,m)=1.0-\mathrm{e}^{-n-m}\sum_{N=1}^{+\infty}\left(\frac{m^N}{N\,!}\sum_{k=0}^{N-1}\frac{n^k}{k\,!}\right),\quad 0\leqslant n\leqslant 2.0, 0\leqslant m\leqslant 2.0$$

本例需要计算两个求和项，其中最外层要求一个无穷级数，分析表达式，可以得知，实际计算中，N 不必取到无穷，只要取到 30 就可以达到较高的精度。本例代码如下：

```
function [m,n,TT] = plot3dnmT(N,L)
% N:inf 的近似,L:[0,2]区间的剖分个数
C = zeros(N,1); % nested - function:Tmn = calcT(mm,nn)中用来存储计算结果
m = linspace(0,2,L);
[m,n] = meshgrid(m,m);
TT = zeros(size(n)); % 和网格数据 m,n 对应的计算出来的 T(m,n)网格数据
for ii = 1:L
    for jj = 1:L
        TT(ii,jj) = calcT(m(ii,jj),n(ii,jj));
    end
end
% ===== 计算 T(m,n)的 nest - function
function Tmn = calcT(mm,nn)
for N1 = 1:N
    C(N1) = (mm^N1/gamma(N1 + 1)) * sum(   nn.^(0:N1 - 1)./gamma(1:N1)   );
    Tmn = 1.0 - exp( - mm - nn) * sum(C);
end
end
mesh(n,m,TT);
end
```

运行下面代码：

```
[m,n,TT] = plot3dnmT(30,30);
```

得到如图 5.2 所示图形。

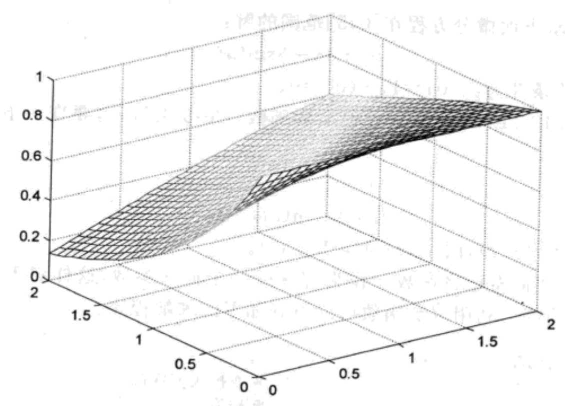


图 5.2　$T(n,m)$的图形

5.4.4　嵌套函数表示待优化的目标函数

本小节以一个有代表性的例子来说明嵌套函数在表示待优化的目标函数方面的应用。

【例 5.4-4】　已知 $w=\left[\frac{\pi}{2},\pi,\frac{3\pi}{2}\right]$；$N=\left[\frac{\pi}{2}-1,-2,-\frac{3\pi}{2}-1\right]$。求下面表达式的最小值：

$$y=\left(\int_0^{w(1)} t^m\cos(t)\mathrm{d}t-N(1)\right)^2+\left(\int_0^{w(2)} t^m\cos(t)\mathrm{d}t-N(2)\right)^2+\left(\int_0^{w(3)} t^m\cos(t)\mathrm{d}t-N(3)\right)^2$$

其中，m 在[0,2]范围内。

代码如下：

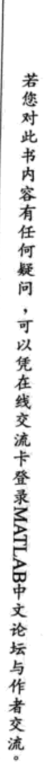
```
function m = Findm
w = [pi/2,pi,pi * 1.5];
N = [pi/2 - 1, - 2, - 1.5 * pi - 1];
function y = ObjecFun(m)
    y = (quadl(@(t) t.^m. * cos(t),0,w(1)) - N(1))^2 + (quadl(@(t) t.^m. * cos(t),0,...
        w(2)) - N(2))^2 + (quadl(@(t) t.^m. * cos(t),0,w(3)) - N(3))^2;
end
m = fminbnd(@ ObjecFun,0,2);
end
```

上述代码中，目标函数即 y 用嵌套函数 ObjecFun 来表示。fminbnd 第一个输入参数是 ObjecFun 的句柄，第二、第三个参数是求解最小值的范围。运行后得到结果如下：

```
>> format long
>> m = Findm
m =
    1.000000256506471
```

5.4.5　嵌套函数在表示微分方程方面的应用

嵌套函数可以方便地表示微分方程，从而可以方便地被 MATLAB 中求解微分方程的函数来调用。请看下例：



【例 5.4-5】 求下面微分方程在[0,5]范围的解：

$$y''+4y=3\sin(at)$$

其中，a 是参数，初始条件为：$y(0)=1, y'(0)=0$。

本例要想应用 MATLAB 的微分方程求解函数 ode45 求解，需要变一下形式。即变成一阶微分方程组的形式：

$$\begin{cases} y_1'=y_2 \\ y_2'=3\sin(at)-4y_1 \end{cases}$$

其中，$y_1(t)$对应于函数 $y(t)$，而 $y_2(t)$对应于 $y'(t)$。

在实际应用中，我们希望将参数 a 作为程序的一个输入参数，这样对于指定不同的 a，就能得到对应的不同的解。利用嵌套函数可以得到如下的求解程序：

```
function example545(a)
tspan = [0,5];                                    % 变量求解区间
y0 = [1,0];                                       % 初值
[t,y] = ode45(@tfys,tspan,y0);                    % 调用 ode45 求解方程
figure;
plot(t,y(:,1),'k-');                              % 画函数 y(t)的曲线
hold on;
plot(t,y(:,2),'k:');                              % 画函数 y(t)导数的曲线
set(gca,'fontsize',12);                           % 设置当前坐标轴字体大小
xlabel('\itt','fontsize',16);                     % 标注 x 轴
ylabel('\ity','fontsize',16);                     % 标注 y 轴
 % 用嵌套函数定义微分方程组
    function dy = tfys(t,y)
        dy(1,1) = y(2);                           % 对应于例子中方程组第一个方程
        dy(2,1) = 3 * sin(a * t) - 4 * y(1);      % 对应于例子中方程组第二个方程
    end
end
```

譬如，当 $a=6$ 时，运行 example545(6)得到如图 5.3 所示结果图。

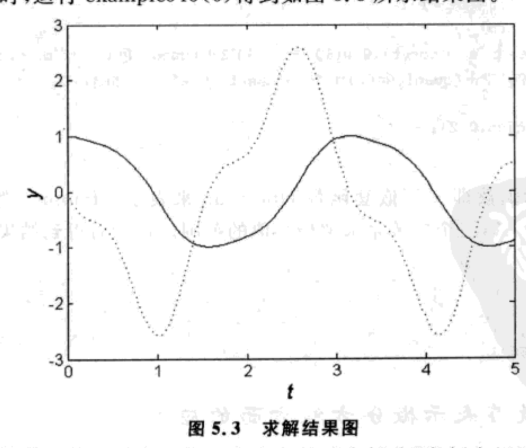


图 5.3 求解结果图

有关嵌套函数的进一步应用介绍将会在本书的后半部分“案例介绍”中穿插进行。这里限于篇幅就不做进一步讨论了。

第二部分

案 例 介 绍





第6章

积分以及积分方程案例

本章将以一些案例来展示 MATLAB 在求积分以及积分方程方面的应用。这些例子很多都是在各技术论坛被网友频繁提问的一些问题。

6.1 案例1:一般区域二重、三重积分 MATLAB 计算方法

6.1.1 概 要

这里讨论的计算方法指的是利用现有的 MATLAB 函数来求解,而不是根据具体的数值计算方法来编写相应程序。对于一般区域上的二重积分,低版本的 MATLAB 向来支持不好,MATLAB 7.X 版本之前不能通过简单的形式直接求取一般区域上的二重积分,往往要借助广泛使用的 NIT 工具箱(数值积分工具箱)来实现。后来到了 MATLAB 7.X 版本,MATLAB引入了匿名函数结构,dblquad 的被积函数可以是匿名函数的形式,利用匿名函数结构,通过适当的改写被积函数,dblquad 可以求解一般区域上的二重积分。其实这种改写就是将积分区域表示成逻辑表达式形式并与被积函数相乘,本质上还是计算的矩形区域的积分。dblquad 的帮助文档给出了用 dblquad 求一般区域上二重积分的简单例子。

仔细分析 dblquad 的帮助文档中求一般区域二重积分的例子就会发现,这种办法是把原被积函数外推到了一个矩形区域上来实现的,该矩形区域包含了被积区域,在被积区域上,外推函数取值和原函数一样,而在矩形区域内被积区域外的那部分,外推函数取值都为 0。这样外推函数在矩形区域上积分的结果就等于原被积函数在被积区域的结果。这种数学上简单的变换导致的结果就是,按这种办法计算量大增,特别是被积区域非常不规则并布满整个矩形区域的时候。

令人高兴的是,从 MATLAB R2009a 版本起,MATLAB 终于有了专门求解一般区域二重积分的函数——quad2d,该函数采用的自适应积分算法基于 L. F. Shampine 的文章:"Vectorized Adaptive Quadrature in MATLAB"(Journal of Computational and Applied Mathematics, 2008)。该函数求解一般区域二重积分的效率要远高于上面提到的 dblquad 的方法。关于它们的比较请参考 6.1.2 节相关部分。

虽然 quad2d 可以求解一般区域二重积分,但还是不能直接求解一般区域三重积分,而 NIT 工具箱也没有一般区域三重积分的计算函数。

本案例的目的是介绍一种在 MATLAB 7.X 版本 MATLAB(不限于 R2009a 版本)里求解一般区域二重、三重积分的思路方法。

6.1.2 一般区域二重积分的计算

在开始介绍方法前,先看一下 dblquad 和 quad2d 函数是怎样求解一般区域的二重积分的。请看下面的例子:

【例 6.1-1】 求被积函数 $f(x,y)=\sqrt{10000-x^2}$ 在 $x^2+y^2\leqslant 10000$ 区域内的积分。

采用 dblquad 的方法，可以写出如下求解代码：

```
tic,y1 = dblquad(@(x,y) sqrt(10^4 - x.^2). * (x.^2 + y.^2 <= 10^4),...
 - 100,100, - 100,100 ),toc
```

运行结果如下：

```
y1 =
    2.6667e + 006
Elapsed time is 8.326637 seconds.
```

采用 quad2d 函数，则代码如下：

```
tic,y2 = quad2d(@(x,y) sqrt(10^4 - x.^2), - 100,100,...
@(x) - sqrt(10^4 - x.^2),@(x)sqrt(10^4 - x.^2)),toc
```

运行结果如下：

```
y2 =
    2.6667e + 006
Elapsed time is 0.024370 seconds.
```

可见 quad2d 的运行效率远远高于 dblquad。

从例 6.1-1 可以看到，用 dblquad 求解一般区域二重、三重积分的思路方法，就是将被积函数"延拓"到矩形或者长方体区域，但是这种方法不可避免引入很多乘 0 运算浪费时间。而 quad2d 虽然运行效率很高，但是 MATLAB R2009a 之前没有这个函数。因此，要想在 MATLAB 2009a 之前的 MATLAB 7 版里计算一般区域二重、三重积分，就需要开辟新的方法。新的方法是调用已有的 MATLAB 函数求解，在求一般区域二重积分时，运行效率和 quad2d 相比有一些差距，但是相对于"延拓"函数的做法，运行效率大大提高了。下面结合一些简单例子来说明计算方法。

【例 6.1-2】 计算下列二重积分：

$$\int_1^2\int_{\sin(x)}^{\cos(x)} xy\,\mathrm{d}y\mathrm{d}x$$

这个积分可以很容易用符号积分算出结果：

```
syms x y
int(int(x * y,y,sin(x),cos(x)),1,2)
ans =
cos(4)/8 - cos(2)/8 - sin(2)/4 + sin(4)/2
>> vpa(ans,20)
ans =
- 0.63541270239994324049
```

上述结果可供稍后的数值计算参考。

如果读者用的是 MATLAB 2009a,可以用

```
quad2d(@(x,y) x. * y,1,2,@(x)sin(x),@(x)cos(x),'AbsTol',1e-12)
```

得到上述结果。

如果用的不是 MATLAB 2009a,那么可以利用 NIT 工具箱里的 quad2dggen 函数。如果既没有 NIT 工具箱,用的也不是 MATLAB 2009a,该怎么办呢?答案是可以利用两次 quadl 函数,注意到 quadl 函数要求积分表达式必须写成向量化形式,所以构造的函数必须能接受向量输入。代码如下:

```
function IntDemo
    function f1 = myfun1(x)
        f1 = zeros(size(x));
        for k = 1:length(x)
            f1(k) = quadl(@(y) x(k) * y,sin(x(k)),cos(x(k)));
        end
    end
y = quadl(@myfun1,1,2)
end
```

myfun1 函数就是构造的原始被积函数对 y 积分后的函数,这时候是关于 x 的函数,要能接受向量形式的 x 输入,所以构造这个函数的时候考虑到 x 是向量的情况。利用 arrayfun 函数(arrayfun 函数的用法读者可以参考 2.4.2 节)可以将 IntDemo 函数精简成一句代码:

```
quadl(@(x) arrayfun(@(xx) quadl(@(y) xx * y,sin(xx),cos(xx)),x),1,2)
```

上面这行代码体现了用 MATLAB 7.X 求一般区域二重积分的一般方法。可以这么理解这句代码:首先

```
@(x) arrayfun(@(xx) quadl(@(y) xx * y,sin(xx),cos(xx)),x)
```

定义了一个关于 x 的匿名函数,供 quadl 调用求最外重(x 从 1 到 2 的)积分,这时候,x 对于

```
arrayfun(@(xx) quadl(@(y) xx * y,sin(xx),cos(xx)),x)
```

就是已知的了,而

```
@(xx) quadl(@(y) xx * y,sin(xx),cos(xx))
```

定义的是对于给定的 xx,求 xx * y 关于 y 的积分函数,这就相当于数学上积完第一重 y 的积分后得到一个关于 xx 的函数,而

```
arrayfun(@(xx) quadl(@(y) xx * y,sin(xx),cos(xx)),x)
```

只是对

```
@(xx) quadl(@(y) xx * y,sin(xx),cos(xx))
```

加了一个循环的壳，保证"积完第一重 y 的积分后得到一个关于 xx 的函数"能够接受向量化的 xx 的输入，从而能够被 quadl 调用。有了这个模板，就可以方便求其他一般积分区域（上下限是函数）形式的二重积分，例如：

【例 6.1-3】 求下列二重积分：

$$\int_{10}^{20}\int_{5x}^{x^2} e^{\sin(x)}\ln(y)\,dy\,dx$$

用 quad2d 函数，上面介绍的方法，以及 dblquad 帮助文档里给的延拓函数的方法得到的运算结果以及运行时间情况如下：

```
tic,y1 = quad2d(@(x,y) exp(sin(x)). * log(y),10,20,@(x)5 * x,@(x)x.^2),toc
tic,y2 = quadl(@(x) arrayfun(@(x) quadl(@(y) exp(sin(x)). * log(y),...
5 * x,x.^2),x),10,20),toc
tic,y3 = dblquad(@(x,y) exp(sin(x)). * log(y). * (y>= 5 * x & y<= x.^2),10,20,50,400),toc
y1 =
    9.368671342614414e + 003
Elapsed time is 0.021152 seconds.
y2 =
    9.368671342161189e + 003
Elapsed time is 0.276614 seconds.
y3 =
    9.368671498376889e + 003
Elapsed time is 1.674544 seconds.
```

由此可以看出，本节介绍的求一般区域二重积分的方法在 MATLAB 2009a 以外的版本中不失为一种好方法，效率要明显高于 dblquad 帮助文档里推荐的做法。更重要的是，这给求解一般区域三重积分提供了一种途径。接下来的章节，将介绍求一般区域三重积分的方法。

6.1.3 一般区域三重积分的计算

6.1.2 节给出了一般区域二重积分计算的例子：

```
quadl(@(x) arrayfun(@(xx) quadl(@(y) xx * y,sin(xx),cos(xx)),x),1,2)
```

可以把其写成模板，即

```
quadl(@(x) arrayfun(@(xx) quadl(@(y) 被积二元函数 f(xx,y),y 的积分下限表达式 g1(xx),y 的积分上限表达式 g2(xx)),x),x 积分下限值,x 积分上限值)
```

现在来看一般区域三重积分的求法，有两种思路：一种是 quadl + quad2d 函数，这需要 MATLAB 2009a 来支持；另一种是用三个 quadl 函数。前者还可细分成先 quad2d 后 quadl，以及先 quadl 后 quad2d。可以得到三种模板，同时结合下面简单的三重积分为例说明：

$$\int_{1}^{2}\int_{x}^{2x}\int_{xy}^{2xy} xyz\,dz\,dy\,dx$$

模板与相应实例如下：

模板1：

```
quadl(@(x) arrayfun(@(xx) quad2d(被积函数 f(xx,y,z)关于 y,z 变量的函数句柄,y 积分下限 y1(xx),y 积分上限 y2(xx),z 积分下限 z1(xx,y),z 积分上限 z2(xx,y)),x),x 积分下限值,x 积分上限值)
```

实例：

```
quadl(@(x) arrayfun(@(xx) quad2d(@(y,z) xx.*y.*z,xx,2*xx,@(y) xx*y,@(y)...
2*xx*y),x),1,2)
```

模板2：

```
quad2d(@(x,y) arrayfun(@(xx,yy) quadl(被积函数 f(xx,yy,z)关于 z 变量的函数句柄,z 积分下限 z1(xx,yy),z 积分上限 z2(xx,yy)),x,y),x 积分下限值,x 积分上限值,y 积分下限 y1(x),y 积分上限 y2(x))
```

实例：

```
quad2d(@(x,y) arrayfun(@(xx,yy) quadl(@(z) xx.*yy.*z,...
xx*yy,2*xx*yy),x,y),1,2,@(x)x,@(x)2*x)
```

模板3：

```
quadl(@(x) arrayfun(@(xx) quadl(@(y) arrayfun(@(yy) quadl(被积函数 f(xx,yy,z)关于 z 变量的函数句柄,z 积分下限 z1(xx,yy),z 积分上限 z2(xx,yy)),y),y 积分下限 y1(xx),y 积分上限 y2(xx)),x),x 积分下限值,x 积分上限值)
```

实例：

```
quadl(@(x) arrayfun(@(xx) quadl(@(y) arrayfun(@(yy) quadl(@(z)...
xx.*yy.*z,xx*yy,2*xx*yy),y),xx,2*xx),x),1,2)
```

模板使用说明：x,y,z 是积分变量，模板中除了用语言描述的参量用相应表达式替换掉外，其余结构保持不变。读者可以运行这三个实例，都比用 triplequad 拓展函数法快得多，而且用 triplequad 拓展函数得到的结果还不精确，其结果如下：

```
tic;quadl(@(x) arrayfun(@(xx) quad2d(@(y,z) xx.*y.*z,xx,2*xx,@(y) xx*y,@(y)...
  2*xx*y),x),1,2),toc
tic;quad2d(@(x,y) arrayfun(@(xx,yy) quadl(@(z)...
  xx.*yy.*z,xx*yy,2*xx*yy),x,y),1,2,@(x)x,@(x)2*x),toc
tic;quadl(@(x) arrayfun(@(xx) quadl(@(y) arrayfun(@(yy) quadl(@(z)...
xx.*yy.*z,xx*yy,2*xx*yy),y),xx,2*xx),x),1,2),toc
tic;triplequad(@(x,y,z) x.*y.*z.*(z<=2*x.*y&z>=x.*y&y<=2*x&y>=x),...
1,2,1,4,1,16),toc
ans =
  179.2969
Elapsed time is 0.037453 seconds.
ans =
  179.2969
Elapsed time is 0.223533 seconds.
ans =
  179.2969
```

若您对此书内容有任何疑问，可以凭在线交流卡登录MATLAB中文论坛与作者交流。

```
Elapsed time is 0.090477 seconds.
ans =
   178.9301
Elapsed time is 78.421721 seconds.
```

从上面代码可以看出，如果读者用的是 MATLAB 2009a，那么计算一般区域三重积分时可以用模板1，MATLAB 2009a 以前的版本(限于7.X版本)可以用模板3，而模板2效率比较低(更加频繁调用函数增加了系统开销)。以上二重、三重积分模板还可以推广应用范围，譬如计算如下积分：

$$\int_{10}^{100}\left(\frac{1}{\int_{x}^{x^2} y\,\mathrm{d}y}\right)\mathrm{d}x$$

就不能套用 dblquad 或者 quad2d，但是用一般二重积分模板稍作变形，可以利用下面代码求解：

```
quadl(@(x) 1./arrayfun(@(xx)quadl(@(y)y,xx,xx^2),x),10,100)
```

对于上述方法的应用，再看一个例子：

【例 6.1-4】 求下列二重积分：

$$\int_{0.2}^{1} 2y\mathrm{e}^{-y^2}\left(\int_{-1}^{1}\frac{\mathrm{e}^{-x^2}}{y^2+x^2}\mathrm{d}x\right)^2\mathrm{d}y$$

上述积分不是简单的二重积分，不能用 dblquad 求解，只能分两次积分求解。利用 quadl 和 arrayfun，可以写出求解代码如下：

```
f1 = quadl(@(y) 2 * y. * exp( - y.^2). * arrayfun(@(y)quadl(@(x) exp( - x.^2)./...
(y.^2 + x.^2), - 1,1),y).^2,0.2,1)
f1 =
    10.2135
```

6.2 案例2：被积函数含有积分项的一类积分的一些求解方法

这里说的含有积分项的被积函数是以下面这个积分为例：

$$\int_{5}^{\mathrm{inf_v}}\int_{5}^{\mathrm{inf_u}}\frac{f_real^2+f_img^2}{\sqrt{u^2+v^2}}\mathrm{d}u\mathrm{d}v$$

其中：

$$f_real=\int_{0}^{b}\int_{0}^{a}(1-\cos(kmx))(1-\cos(kny))\cos(ux+vy)\mathrm{d}x\mathrm{d}y$$

$$f_img=\int_{0}^{b}\int_{0}^{a}(1-\cos(kmx))(1-\cos(kny))\sin(ux+vy)\mathrm{d}x\mathrm{d}y$$

从上面的公式，可以看到原始被积函数的分子是两个二重积分的平方和，整体来看，上述计算量相当于一个四重积分，因此计算量会很大。

本节将介绍一系列的求解方法，这些方法大体按求解效果好坏以及运行效率高低排列。

首先最容易想到的是网格法，下一小节将以网格法为例，来展示网格法的具体步骤。

6.2.1　网格求解法

这种方法的思路是给定 u、v 的积分上限 inf_u、inf_v，将积分区域剖分成 nn×nn 个小正方形，每个小正方形的面积乘以被积函数在正方形中心的值作为被积函数在该小区域积分的近似值。最后在对所有的这些区域求和得到最后的积分。nn 越大，分的越细，精度越高，计算时间越长。以下是整个过程代码：

【例 6.2-1】　划分网格法示例：

```
function INT = example6_2_1(inf_u,inf_v,nn)
 % inf_u,u 无穷上限的近似
 % inf_v,v 无穷上限的近似，
 % nn,u,v 积分区间的等分个数

U = linspace(0,inf_u,nn + 1);
V = linspace(0,inf_u,nn + 1);
int = zeros(nn);
for ku = 1:nn
    for kv = 1:nn
        int(ku,kv) = Tf_Integrand( (U(ku) + U(ku + 1))/2, (V(kv) + V(kv + 1))/2 ) * ...
            (inf_u/nn) * (inf_v/nn);
    end
end
INT = sum(int(:));

    function z = Tf_Integrand(u,v)
        m = 1;
        n = 1;
        a = 1;
        b = 1;
        km = 2 * m * pi/a;
        kn = 2 * n * pi/b;

        f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
        f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
        tmp_rel = dblquad(f_rel, 0,a, 0,b);
        tmp_img = dblquad(f_img, 0,a, 0,b);

        z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
    end
end
```

运行得到如下结果：

```
tic;INT = example6_2_1(40,40,15),toc
INT =
    6.3876
Elapsed time is 31.555133 seconds.
```

【说明】 上述方法将积分区域划分成均匀的 $n\times n$ 个网格，以每个网格中心的函数值乘以网格的面积作为函数在这个网格上积分的近似值，由于内部函数还要计算两个二重积分的平方，因此这一方法要计算 $2n^2$ 个二重积分的平方。为了保证精度，n 通常会较大，所以计算量还是非常大的。

下面介绍另一种方法，基于插值的方法。

6.2.2 插值求解法

本方法是将插值法和均匀划分网格法结合起来。在求类似上面那样复杂，计算量大但连续性、光滑性比较好的函数在一个区域上的积分时，首先粗划分积分区域（生成比较粗的网格），计算函数在粗网格节点上的值，然后在上面网格基础上细划分网格，根据粗网格节点上的函数值利用 interp2 或者 griddata 函数插值生成细网格节点上的函数值，最后再乘以细网格面积并求和得到整个函数在积分区域上的近似值。以下是以前面提到的的积分为例子的代码：

【例 6.2-2】 插值求解法示例：

```
function [INT,TfI] = example6_2_2(inf_u,inf_v,mm,nn)
% inf_u,u无穷上限的近似
% inf_v,v无穷上限的近似，
% mm,u,v积分限细等分个数(目的是插值生成足够多的Tf_Integrand函数点)
% nn,u,v积分限粗等分个数(目的是生成插值的骨架点)
% 细网格数据
[UI,VI] = meshgrid( (inf_u/mm:inf_u/mm:inf_u),(inf_v/mm:inf_v/mm:inf_v)) ;
% 粗网格数据
[U,V] = meshgrid( linspace(inf_u/mm,inf_u,nn),linspace(inf_v/mm,inf_v,nn)) ;
Tf = zeros(nn);
for k = 1:nn^2
    Tf(k) = Tf_Integrand( U(k), V(k) ); % 生成粗网格节点函数值
end
% 插值并求近似积分
TfI = interp2(U,V,Tf,UI,VI);
INT = TfI * inf_u * inf_v/(mm^2);
INT = sum(INT(:));
    function z = Tf_Integrand(u,v)
        m = 1;
        n = 1;
        a = 1;
        b = 1;
        km = 2 * m * pi/a;
        kn = 2 * n * pi/b;
        f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
        f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
        tmp_rel = dblquad(f_rel, 0,a, 0,b);
        tmp_img = dblquad(f_img, 0,a, 0,b);
        z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
    end
end
```



与 6.2.1 节的方法对比如下：

划分网格法：分成 25×25 个网格，计算 1 250 个二重积分的平方，所用时间和结果：

```
tic,INT = example6_2_1(40,40,25);toc
Elapsed time is 85.669888 seconds.
INT =
    6.8267
```

插值法：粗分成 15×15 个网格，计算 450 个二重积分的平方，然后根据这 225 个函数点插值生成 510×50 个函数值，最后计算近似积分，所用时间和结果：

```
tic;[INTi,TfI] = example6_2_2(40,40,50,15);toc
Elapsed time is 30.476624 seconds.
INTi =
    6.0241
```

【说明】 通过对比可以看到，插值法比划分网格法运算速度有了提高，但是这种插值没有利用原始函数的梯度信息，是均匀插值，这必然使得计算精度受到影响。为此，需要寻求一种保证精度和速度的方法。

下面介绍第三种方法，基于 RBF 神经网络的方法。

6.2.3 RBF 神经网络逼近法

RBF 神经网络(径向基神经网络)是人工神经网络的一种，有“局部感知场”的美誉，可以任意逼近光滑性比较好的函数。下面是一个用 RBF 网络对 MATLAB 里 peaks 这个二维函数逼近的例子，读者可以比较一下它与插值法的逼近精度。

【例 6.2-3】 RBF 逼近 peaks 函数。

在命令窗口运行：

```
peaks
```

命令，可以得到图 6.1 所示的 peaks 函数图。下面代码可以比较 RBF 神经网络逼近 peaks 函数的精度和插值法逼近 peaks 函数的精度：

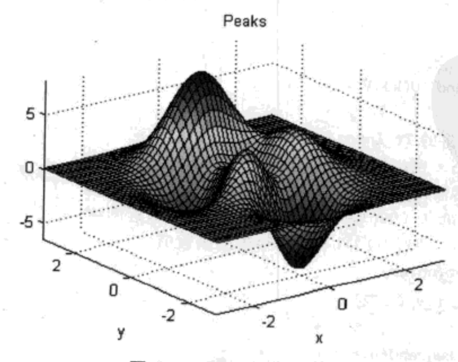


图 6.1　peaks 函数图像

```
[X,Y] = meshgrid( - 3:.25:3);
Z = peaks(X,Y); % 生成网络训练样本
input = [X( : )';Y(:)']; % 网络输入
output = Z( : )'; % 网络输出
net = newrb(input,output,0.00001,0.5); % 生成 peaks 的逼近网络
[XI,YI] = meshgrid( - 3 : .125 : 3);
Znet = sim(net,[XI( : )';YI( : )']); % 网络给出的 peaks 函数值
Znet = reshape(Znet,49,49);
Zpeak = peaks(XI,YI); % 真实的 peaks 函数值
err1 = abs(Znet - Zpeak); % 网络逼近值和真实值的绝对误差
Zinter = interp2(X,Y,Z,XI,YI); % 插值给出的 peaks 函数值
err2 = abs(Zinter - Zpeak); % 插值逼近值和真实值的绝对误差
disp(['RBF 网络逼近的绝对误差和为:',num2str(sum(err1( : )))])
disp(['插值逼近的绝对误差和为:',num2str(sum(err2( : )))])
```

运行上述代码，得到如下结果：

```
RBF 网络逼近的绝对误差和为:5.7598
插值逼近的绝对误差和为:66.0807
```

可见，RBF 神经网络的逼近能力大大高于插值法的逼近能力。如果利用 RBF 神经网络来逼近本案例开始提出的那个复杂积分，那么可以比插值法提高精度，同时速度不至于受太大影响。相应的代码如下：

【例 6.2-4】 RBF 神经网络逼近法。

```
function [INT,TfInet] = rbf_calc(inf_u,inf_v,nn)
% inf_u,u 无穷上限的近似
% inf_v,v 无穷上限的近似，
% nn,u,v 积分限区间的等分个数
[UI,VI] = meshgrid(linspace(0,inf_u,nn + 1),linspace(0,inf_v,nn + 1));
UI = (UI(1 : end - 1,1 : end - 1) + UI(2 : end,2 : end))/2;
VI = (VI(1 : end - 1,1 : end - 1) + VI(2 : end,2 : end))/2; % 生成 nn * nn 网格的中心点
[U,V] = meshgrid(linspace(UI(1,1),UI(1,end),nn/1.5),linspace(VI(1,1),VI(end,1),...
    nn/1.5)); % 生成网络训练样本网格，样本网格均匀分布在 nn * nn 网格区域内，密度为 UI,VI
% 网格的 1/1.5
Tf = zeros(size(U));
for k = 1:length(Tf)^2
    Tf(k) = Tf_Integrand( U(k),V(k));
end
% 训练网络并给出足够多的 Tf_Integrand 函数点并计算积分
input = [U( : )';V( : )']; % 网络输入
output = Tf( : )'; % 网络输出
net = newrb(input,output,0.00001,3 * inf_u/nn); % 生成逼近网络
TfInet = sim(net,[UI( : )';VI( : )']); % 网络给出的函数值
TfInet = reshape(TfInet,nn,nn);
INT = TfInet * inf_u * inf_v/(nn^2);
INT = sum(INT( : ));
    function z = Tf_Integrand(u,v)
```

```
            m = 1;
            n = 1;
            a = 1;
            b = 1;
            km = 2 * m * pi/a;
            kn = 2 * n * pi/b;
            f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
            f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
            tmp_rel = dblquad(f_rel, 0,a, 0,b);
            tmp_img = dblquad(f_img, 0,a, 0,b);
            z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
        end
    end
```

以下是 RBF 神经网络逼近法同第一种网格划分法的对比：

```
tic,INT = example6_2_1(40,40,30);toc
Elapsed time is 114.169028seconds.
>> INT
INT =
    6.9354
>> tic,[INTnet,TfInet] = rbf_calc(40,40,30);toc
NEWRB, neurons = 0, MSE = 0.00315257
NEWRB, neurons = 25, MSE = 0.000117715
NEWRB, neurons = 50, MSE = 1.82648e - 005
Elapsed time is 50.255045 seconds.
>> INTnet
INTnet =
    7.0136
```

从对比结果可以看出 RBF 神经网络逼近法从精度和速度上都兼顾了！事实上，训练好网络后，完全可以更加多地细分网格，因为 net 是 Tf_Integrand 函数的近似，给定一组特定的(u，v)，用 sim(net,[u;v])可以非常迅速求出 Tf_Integrand 函数的近似值。这样在求最后的结果时就不用担心网格太密的情况下计算量大了。

【说明】 被积函数 Tf_Integrand 是关于 u，v 的二元函数，只不过这个函数含有两个二重积分的平方，很复杂。因此，上述三种方法都是基于积分的基本定义来计算的，即细分积分区域，每个小积分区域的面积乘以该区域上的函数值(面积足够小的话可以认为该区域函数值与面积一致)然后再求和。第一种方法的思想就是，网格化积分区域后直接计算 Tf_Integrand 函数在各个网格中心的函数值作为该网格区域的函数值，然后乘以各个网格面积并求和作为积分值的近似。因此这种方法随着积分区域的扩大，如果保证精度的话计算量会非常大。

第二种方法是用插值的方法去近似被积函数 Tf_Integrand，只需要计算出 Tf_Integrand 在一些格点上的值，剩下的格点上的函数值插值得出。这种方法精度很有限。

第三种方法是用 RBF 神经网络去近似被积函数 Tf_Integrand，也只需给出 Tf_Integrand 在一些格点上的值，剩下的格点上的函数值由近似网络给出。RBF 神经网络逼近函数的能力很高，因此，该方法优于第二种方法。

各种数值积分的基本思想都是用简单函数近似代替被积函数，所谓简单，就是能够非常方便快速地求出函数值。原题中 Tf_Integrand 这个函数很复杂，之所以复杂是因为计算量很大，为了得到(u,v)上的函数值，要付出计算两个二重积分的平方的代价！这比那些由初等函数复合而成的函数复杂得多。

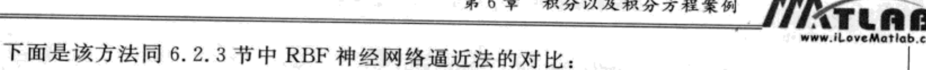
现在第三种方法已经用 RBF 神经网络逼近原函数了，接下来就是如何利用逼近好的网络采取有效的积分方法（不是简单地用面积乘以函数值）来计算积分。因此可以用 dblquad 函数调用网络来实现，这就是第四种方法。

6.2.4 dblquad 调用 RBF 神经网络法

这种方法是将训练好的 RBF 神经网络作为被积函数的近似函数来调用。具体实现见下面例子：

【例 6.2-5】 dblquad 函数调用 RBF 神经网络法：

```
function INT = rbf_dblquad(inf_u,inf_v,nn)
% inf_u,u 无穷上限的近似
% inf_v,v 无穷上限的近似，
% nn,u,v 积分限区间的等分个数
[UI,VI] = meshgrid(linspace(0,inf_u,nn + 1),linspace(0,inf_v,nn + 1));
UI = (UI(1 : end - 1,1 : end - 1) + UI(2 : end,2 : end))/2;
VI = (VI(1 : end - 1,1 : end - 1) + VI(2 : end,2 : end))/2; % 生成 nn * nn 网格的中心点
[U,V] = meshgrid(linspace(UI(1,1),UI(1,end),nn/1.5),linspace(VI(1,1),VI(end,1),...
    nn/1.5)); % 生成网络训练样本网格，样本网格均匀分布在 nn * nn 网格区域内，密度为 UI,VI
% 网格的 1/1.5
Tf = zeros(size(U));
for k = 1 : length(Tf)^2
    Tf(k) = Tf_Integrand( U(k),V(k));
end
% 训练网络并给出足够多的 Tf_Integrand 函数点并计算积分
input = [U( : )';V( : )']; % 网络输入
output = Tf( : )'; % 网络输出
net = newrb(input,output,0.00001,3 * inf_u/nn); % 生成逼近网络
INT = dblquad(@(u,v) sim(net,[u( : )';v * ones(size(u( : )'))]),0,inf_u,0,inf_v);
    function z = Tf_Integrand(u,v)
        m = 1;
        n = 1;
        a = 1;
        b = 1;
        km = 2 * m * pi/a;
        kn = 2 * n * pi/b;
        f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
        f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
        tmp_rel = dblquad(f_rel, 0,a, 0,b);
        tmp_img = dblquad(f_img, 0,a, 0,b);
        z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
    end
end
```



下面是该方法同 6.2.3 节中 RBF 神经网络逼近法的对比：

```
tic,[INTnet,TfInet] = rbf_calc(40,40,30);toc
NEWRB, neurons = 0, MSE = 0.00315257
NEWRB, neurons = 25, MSE = 0.000117715
NEWRB, neurons = 50, MSE = 1.82648e - 005
Elapsed time is 49.841527 seconds.
>> INTnet
INTnet =
    7.0517

tic,INT = rbf_dblquad(40,40,15);INT,toc
NEWRB, neurons = 0, MSE = 0.0022636
NEWRB, neurons = 25, MSE = 2.70421e - 005
INT =
    6.9604
Elapsed time is 40.577387 seconds.
```

【说明】　需要注意的是，Tf_Integrand 函数在(0,0)点是奇异的，不能选取(0,0)点作为训练样本，限于神经网络外推能力的有限性，在(0,0)点附近，神经网络法给出的值存在误差，这些误差会造成最后的积分误差。

到目前为止，上述方法都是通过拟合或者逼近原函数的方法来实现积分。由于选取的点有限，所以不可避免地造成了计算结果偏离真实值较远。能否直接从原始函数出发来求解呢？即通过现有的 MATLAB 求积分的各种函数调用来实现原积分问题的求解呢？答案是肯定的，下面即将给出的两种方法就是这样做的。

6.2.5　dblquad+arrayfun 方法

参考 6.1 节中介绍的求一般区域积分的方法可以得到第五种方法，即 dblquad 函数+arrayfun 函数的方法。通过设置外层 dblquad 和内层 dblquad 的精度以及 arrayfun 函数精简代码，速度和精度又较之第四种方法有了提高。代码如下：

【例 6.2-6】　dblquad 函数+arrayfun 函数的方法：

```
function fx = CalcInteDemo(OutTol,InTol)
% OutTol,外层 dblquad 精度
% InTol,内层 dblquad 精度
m = 1;
n = 1;
a = 1;
b = 1;
km = 2 * m * pi/a;
kn = 2 * n * pi/b;

    function z = Tf_Integrand(u,v)
        f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
        f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
```

```
        tmp_rel = dblquad(f_rel, 0,a, 0,b,InTol);
        tmp_img = dblquad(f_img, 0,a, 0,b,InTol);
        z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
    end

fx = dblquad(@(u,v) arrayfun(@(u) Tf_Integrand(u,v) ,u),0, 40, 0, 40,OutTol);
end
```

下面给出和第四种方法也就是6.2.4节中介绍的方法的对比：

```
format long
tic,INT = rbf_dblquad(40,40,15);INT,toc
NEWRB, neurons = 0, MSE = 0.0022636
NEWRB, neurons = 25, MSE = 2.70421e - 005
INT =
    6.960433723093214
Elapsed time is 38.574448 seconds.

tic,INT = rbf_dblquad(40,40,30);INT,toc
NEWRB, neurons = 0, MSE = 0.00315257
NEWRB, neurons = 25, MSE = 0.000117715
NEWRB, neurons = 50, MSE = 1.82648e - 005
INT =
    7.179902533239947
Elapsed time is 100.918145 seconds.

tic;fx = CalcInteDemo(0.005,0.0001),toc
fx =
    7.436377681182572
Elapsed time is 17.580248 seconds.
>> tic;fx = CalcInteDemo(0.01,0.0001),toc
fx =
    7.424836568066218
Elapsed time is 15.335061 seconds.
>> tic;fx = CalcInteDemo(0.01,0.001),toc
fx =
    7.421310504171474
Elapsed time is 9.144342 seconds.
```

从上面的对比可以明显看出，第五种方法无论是从速度上还是从精度上都完全比前面的四种方法要好，在实际应用中是有效的求解方法。

6.2.6 quad2d+arrayfun 方法

第五种方法已经比前面四种方法的精度和速度都要好多了。但分析原积分函数会发现，原始积分在km较大的时候是震荡积分，而dblquad里面默认的是用quad来计算内外重积分，quad的算法是自适应simpson方法，它不大适用于震荡函数积分。

那如何有效求解震荡型的积分呢？如果读者用的是MATLAB R2009a或者更新的版本，

那么可以有两种途径：一维积分用 quadgk，二维积分用 quad2d。

在开始介绍第六种方法之前，先看一下分别用 quadgk，quadl，quad 求震荡函数积分的精度和速度差异：

【例 6.2-7】 比较 quadgk，quadl，quad 求震荡函数积分的精度和速度差异。

```
tic;quadgk(@(x) sin(50 * x). * cos(50 * x),0,20),toc
tic;quadl(@(x) sin(50 * x). * cos(50 * x),0,20),toc
tic;quad(@(x) sin(50 * x). * cos(50 * x),0,20),toc
ans =
     0.006837297745507
Elapsed time is 0.006194 seconds.
Warning: Maximum function count exceeded; singularity likely.
> In quadl at 104
ans =
     2.043885879186874
Elapsed time is 0.097481 seconds.
ans =
     0.006836670520123
Elapsed time is 0.101185 seconds.
```

可见只有 quadgk 计算得又快又准。对于二维震荡函数积分，可以利用 quad2d 来计算，当然也可以利用 quadgk＋arrayfun 来计算（效率不如 quad2d 高，但远比 dblquad 高），请看例 6.2-8：

【例 6.2-8】 比较 dblquad 和 quadgk＋arrayfun，和 quad2d 求二维震荡函数积分的速度和精度差异。

```
tic;dblquad(@(x,y) sin(10 * x). * cos(10 * y),0,5,0,5,1e-8),toc
tic;quadgk(@(x)arrayfun(@(x) quadgk(@(y) sin(10 * x). * cos(10 * y),0,5,'abstol',...
1e-8),x),0,5,'abstol',1e-8),toc
tic;quad2d(@(x,y) sin(10 * x). * cos(10 * y),0,5,0,5,'abstol',1e-8),toc
ans =
     -9.192040105538080e-005
Elapsed time is 2.395602 seconds.
ans =
     -9.192033149052741e-005
Elapsed time is 0.155306 seconds.
ans =
     -9.192035999367503e-005
Elapsed time is 0.097786 seconds.
```

从上面的结果看，quad2d 的效率最高，quadgk 虽然效率不如 quad2d 高，但远远比 dblquad 高。回到本案例开始提出的那个积分上，得到第六种解法代码如下：（当然还可以利用 quadgk 写出第七种，但由于 quad2d 效率很高，这里就不写了，有兴趣的读者可以自己实现。值得说明的是 quad2d 通过设置 'Singular'为 false 还可以求解一些非震荡光滑函数。）

【例 6.2-9】 quad2d＋arrayfun 方法。

```
function fx = CalcInteDemo2(OutTol,InTol)
m = 1;
n = 1;
a = 1;
b = 1;
km = 2 * m * pi/a;
kn = 2 * n * pi/b;
    function z = Tf_Integrand(u,v)
        f_rel = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * cos(u. * x + v. * y);
        f_img = @(x,y) (1 - cos(km * x)). * (1 - cos(kn * y)). * sin(u. * x + v. * y);
        tmp_rel = quad2d(f_rel, 0,a, 0,b,'abstol',InTol);
        tmp_img = quad2d(f_img, 0,a, 0,b,'abstol',InTol);
        z = (tmp_rel.^2 + tmp_img.^2)./sqrt(u.^2 + v.^2);
    end
fx = quad2d(@(u,v) arrayfun(@(u,v) Tf_Integrand(u,v) ,u,v),0, 40, 0, 40,...
    'abstol',OutTol);
end
```

和第五种方法的对比如下：

```
tic;fx = CalcInteDemo2(0.001,0.0001),toc
fx =
    7.475217656951065
Elapsed time is 12.362713 seconds.
>> tic;fx = CalcInteDemo(0.001,0.0001),toc
Warning: Minimum step size reached; singularity possible.
> In quad at 103
  In dblquad > innerintegral at 84
  In quad at 77
  In dblquad at 60
  In CalcInteDemo at 17
fx =
    7.468956689792021
Elapsed time is 24.572359 seconds.
```

可以看出，对于本案例，第六种方法给出的值最可靠，精度最好，速度最快。

6.3 案例3：一般区域 n 重积分

通过上面两个案例，一般区域上的一重、二重、三重积分都已经讨论过了。接下来一个自然的问题就是如何求解一般区域上的 n 重积分？

本案例就是讨论一般区域 n 重积分的计算方法。这里讨论的 n，理论上讲不限大小，但是鉴于 n 重积分的特点，如果积分重数高，同时积分区域范围较大，数值积分计算量十分巨大，如果各重积分原函数都能显式表示，对于多重积分，采用符号积分是第一选择。当只能采用数值积分的时候，很多时候四重、五重积分计算时间都较长。所以虽说不限制 n 的大小，但是能实际计算的 n 重积分，n 一般不会很大。

下面介绍求解一般区域 n 重积分的函数——nIntegrate。需要说明的是 nIntegrate 函数是调用现成的 MATLAB 函数来编写 n 重积分，因此当积分重数高的时候，存在着大量的函数

调用，因此整体运行效率会受到影响。但读者可以从该函数中加深对本章前面两个案例的理解，因为本案例在内容上可以看作是前面两个案例的推广与递进。

该函数的调用格式如下：

```
f = nIntegrate(fun,low,up)
```

f 为函数的返回值，是 n 重积分的积分结果。

fun 是被积函数字符串形式，不同的变量依次以 $x1,x2,\cdots,xn$ 表示（需要注意的是：必须以 $x1,x2,\cdots,xn$ 这种形式表示，像 $y1,y2,\cdots$，yn 或是其他表示方法都不行）。

low 和 up 都是长度为 n 的 cell 数组，low 存储从外到内各重积分的积分下限函数，up 存储从外到内各重积分的积分上限函数（都是字符串形式）。low 和 up 内的函数表示都要遵循一些原则，这些原则在程序注释里进行了说明，后续例子中将进一步详细说明。

下面给出 nIntegrate 函数的代码：

```
function f = nIntegrate(fun,low,up)
 % f，返回值,n 重积分积分结果
 % fun，是被积函数字符串形式；
 % low 存储从外到内各重积分的积分下限函数；
 % up 存储从外到内各重积分的积分上限函数(都是字符串形式)
 % 为了保证函数正常运行,low 和 up 内的函数遵循如下原则:设积分重数为 m,最内层到最外层的
 % 积分变量依次以 xm,...x2,x1 来表示(只能以这样顺序,其他顺序或者别的字母表示变量都不可以)
 % 最内层的上下限函数不管是不是关于 x(m-1)的函数都要求 x(m-1)必须出现,不是其函数时
 % 可以写成“+0*x(m-1)”等形式,依次类推次内层要求 x(m-2)必须要出现等等,一直到次外层
 % 要求变量 x1 出现,最外层才是常数。
N = length(low); % 判断积分重数
fun = vectorize(fun); % 将被积函数写成点乘形式,方便数值积分函数调用
if verLessThan('MATLAB','7.8') % 判断当前运行该函数的 MATLAB 版本是否低于 7.8 即 R2009a
    % 低于 7.8 调用 GenerateExpr_quadl 函数递归构造求解表达式
    expr = GenerateExpr_quadl(N);
else %7.8 或者以后的版本调用 GenerateExpr_quad2d 函数递归构造求解表达式
    if mod(N,2) == 0
        expr = GenerateExpr_quad2d(N);
    else
        expr = ['quadl(@(x1) arrayfun(@(x1)',...
            GenerateExpr_quad2d(N-1),',x1),',low{1},',',up{1},')'];
    end

end
% =======================================================
% 主要利用 quadl 函数递归构造求解表达式,适用于 R2009a 以前的版本
% =======================================================
    function expr = GenerateExpr_quadl(n)
        if n == 1
            expr = ['quadl(@(x',num2str(N),')',fun,',',low{N},',',up{N},')' ];
        else
            expr = ['quadl(@(x',num2str(N-n+1),') arrayfun(@(x',...
```

```
                num2str(N - n + 1),')',GenerateExpr_quadl(n - 1),',x',num2str(N - n + 1),...
                '),',low{N - n + 1},',',up{N - n + 1},')'];
        end
    end
% ==========================================================
% 主要利用 quad2d 函数递归构造求解表达式,适用于 R2009a 以及以后的版本
% ==========================================================
    function expr = GenerateExpr_quad2d(n)
        if n == 2
            expr = ['quad2d(@(x',num2str(N - 1),',x',num2str(N),')',fun,',...
                low{N - 1},',',up{N - 1},',@(x',num2str(N - 1),')',low{N},',@(x',...
                num2str(N - 1),')',up{N},')'];
        else
            expr = ['quad2d(@(x',num2str(N - n + 1),',x',num2str(N - n + 2),')',...
                'arrayfun(@(x',num2str(N - n + 1),',x',num2str(N - n + 2),')',...
                GenerateExpr_quad2d(n - 2),...
                ',x',num2str(N - n + 1),',x',num2str(N - n + 2),'),',...
                low{N - n + 1},',',up{N - n + 1},...
                ',@(x',num2str(N - n + 1),')',low{N - n + 2},...
                ',@(x',num2str(N - n + 1),')',up{N - n + 2},')'];
        end
    end
f = eval(expr);
end
```

函数可运行于 MATLAB 7.0 以后的版本，其中又以 MATLAB R2009a 为界设置了不同的求解方法。如果用的是 MATLAB 2009a，那么速度明显会比 MATLAB 2009a 以前的版本快很多。

现举例说明函数用法。

【例 6.3-1】 计算如下积分：

$$\int_0^1\left(\int_0^1\left(\int_0^1\left(\int_0^1 e^{x_1x_2x_3x_4}\,dx_4\right)dx_3\right)dx_2\right)dx_1$$

上述四重积分，构造输入变量并求解，代码如下：

```
fun = 'exp(x1 * x2 * x3 * x4)';
% 由于各层积分上下限都是常数，为了和程序中要求的保持一致，积分上下限函数可以写成如下形式，
% 当然还可以写成任意满足程序要求的形式，譬如 up = {'1','0 * x1 + 1','0 * (x1 + x2) + 1','0 * (x1 * x2
% - x3) + 1'};等
up = {'1','0 * x1 + 1','0 * x2 + 1','0 * x3 + 1'};
low = {'0','0 * x1','0 * x2', '0 * x3'};
format long
f = nIntegrate(fun,low,up)
f =
    1.069397608859771
% 和真实值比较
syms x1 x2 x3 x4
>> double(int(int(int(int(exp(x1 * x2 * x3 * x4),x4,0,1),x3,0,1),x2,0,1),x1,0,1))
```

```
Warning: Explicit integral could not be found.
> In sym.int at 64
Warning: Explicit integral could not be found.
> In sym.int at 64
Warning: Explicit integral could not be found.
> In sym.int at 64
ans =
    1.069397608859771
```

由此可见，nIntegrate 函数求解出的结果和符号计算得到的结果是一样的。

【例 6.3-2】 计算如下积分：

$$\int_1^2\left(\int_{x_1}^{2x_1}\left(\int_{x_1x_2}^{2x_1x_2}x_1x_2x_3\,\mathrm{d}x_3\right)\mathrm{d}x_2\right)\mathrm{d}x_1$$

对于上述积分，构造输入变量如下：

```
fun = 'x1 * x2 * x3';
up = {'2','2 * x1','2 * x1 * x2'};
low = {'1','x1','x1 * x2'};
```

然后按 f=nIntegrate(fun,low,up)调用即可。积分结果如下：

```
f =
    179.2969
```

【例 6.3-3】 计算如下四重积分：

$$\int_1^2\left(\int_{x_1}^{3x_1}\left(\int_{x_1x_2}^{2x_1x_2}\left(\int_{x_1+x_1x_3}^{x_1+2x_1x_3}\sqrt{x_1x_2}\ln(x_3)+\sin\left(\frac{x_4}{x_2}\right)\mathrm{d}x_4\right)\mathrm{d}x_3\right)\mathrm{d}x_2\right)\mathrm{d}x_1$$

求解上述积分的代码如下：

```
fun4 = 'sqrt(x1 * x2) * log(x3) + sin(x4/x2)' % 构造被积函数字符表达式
up4 = {'2','3 * x1','2 * x1 * x2','x1 + 2 * x1 * x3 + 0 * x2'} % 积分上限函数字符表达式
low4 = {'1','x1','x1 * x2','x1 + x1 * x3 + 0 * x2'} % 积分下限函数字符表达式
f = nIntegrate(fun4,low4,up4)
fun4 =
sqrt(x1 * x2) * log(x3) + sin(x4/x2)
up4 =
    '2'    '3 * x1'    '2 * x1 * x2'    'x1 + 2 * x1 * x3'
low4 =
    '1'    'x1'    'x1 * x2'    'x1 + x1 * x3'
f =
    1.5025e + 003
```

【例 6.3-4】 计算如下五重积分：

$$\int_0^1\mathrm{d}x_1\int_{\exp(x_1)/2}^{\exp(x_1)}\mathrm{d}x_2\int_{(x_1+\sin(x_2))/2}^{x_1+\sin(x_2)}\mathrm{d}x_3\int_{(x_1+x_3)/2}^{x_1+x_3}\mathrm{d}x_4 \rightarrow$$

$$\leftarrow\int_{x_4}^{2x_4}(\sin(x_1\exp(x_2\sqrt{x_3}))+x_4^{x_5})\,\mathrm{d}x_5$$

求解上述积分的代码如下：

```
fun5 = 'sin(x1 * exp(x2 * sqrt(x3))) + x4^x5'
up5 = {'1','exp(x1)','x1 + sin(x2)','x1 + x3','2 * x4'}
low5 = {'0','exp(x1)/2','x1/2 + sin(x2)/2','x1/2 + x3/2','x4'}
f = nIntegrate(fun5,low5,up5),
fun5 =
sin(x1 * exp(x2 * sqrt(x3))) + x4^x5
up5 =
    '1'    'exp(x1)'    'x1 + sin(x2)'    'x1 + x3'    '2 * x4'
low5 =
  Columns 1 through 4
    '0'    'exp(x1)/2'    'x1/2 + sin(x2)/2'    'x1/2 + x3/2'
  Column 5
    'x4'
f =
    5.7981
```

需要说明的是，随着积分重数的增加，大量函数调用随之发生，计算量也会急剧增加，因此，上述求解过程会比较费时间，在现实的计算时间内，能够以比较高的精度求解的积分重数不会太多。但上述介绍的方法在很多时候还是能够方便快捷的求出常见的一些维数不太高的多重积分的解的。

6.4 案例4：蒙特卡洛法计算n重积分

6.4.1 概 述

在6.3节已经看到，对于一些积分重数比较高的积分，如果再按照传统方法进行计算的话，运算量会非常大，求解时间随着重数的增加而迅速增加。6.3节虽然给出了一般区域n重积分的通用程序，但是在实际计算中，n不会很大，一般n超过5后，求解时间就会长到无法接受。对于更高维的积分，一个非常有效的求解方法就是蒙特卡洛积分法。

蒙特卡洛方法又称随机抽样法或统计试验方法，它用于求积分时，与积分重数无关，这一点非常重要。虽然四维以下的积分用蒙特卡洛法效率可能不如传统的一些数值积分方法，但是维数高的时候，蒙特卡洛法比传统方法要有效得多，而且实现起来也非常容易。可以说，计算高维积分是蒙特卡洛方法最成功和典型的应用。

实际应用中，有多种蒙特卡洛方法可以计算n重积分，这里介绍基本的蒙特卡洛法和等分布序列的蒙特卡洛法。

6.4.2 基本的蒙特卡洛积分法

设D为n维空间R^n的一个区域，$f(x)\in D\subset R^n\to R$，区域$D$上的$n$重积分用下式表示：

$$I=\int_D f(p)\mathrm{d}p$$

I可以被认为等于区域D的测度乘以函数f的期望。基本的蒙特卡洛法就是找一个超立方体（测度已知，为M_C）包含区域D，在D内随机生成n（n一般足够大）个均匀分布的点，统计落入

区域 D 的点，假设有 m 个，则区域 D 的测度

$$M_D \approx \frac{mM_C}{n}$$

函数 f 的期望

$$\overline{f} \approx \frac{1}{m}\sum_{i\in D} f(p_i)$$

从而有

$$I \approx M_D\overline{f} = \frac{M_C}{n}\sum_{i\in D} f(p_i)$$

这就是计算重积分的基本的蒙特卡洛法。可以看出，该方法实现起来很方便。下面结合例子来说明该方法。

【例 6.4－1】 用基本的蒙特卡洛法计算例 6.3－1 中的积分：

$$\int_0^1\left(\int_0^1\left(\int_0^1\left(\int_0^1 e^{x_1x_2x_3x_4}\,dx_4\right)dx_3\right)dx_2\right)dx_1$$

这个四重积分在例 6.3－1 中已经用 nIntegrate 函数得到结果为：1.069 397 608 859 771，用基本的蒙特卡洛法求解如下：

```
% 构造被积函数，x 为长为 4 的行向量或者矩阵(列数为 4)。x 的每一行表示四维空间中的一个点
f = @(x) exp(prod(x,2));
n = 10000;
X = rand(n,4); % 随机生成 n 个四维单位超立方体内的点
format long
I = sum(f(X))/n % 用基本的蒙特卡洛法估计积分值
I =
    1.069840982686524
```

求出的结果和 nIntegrate 求出的结果相差不大。

接下来再看不规则区域上的多重积分的蒙特卡洛解法。

【例 6.4－2】 计算如下积分：

$$\int_1^2\left(\int_{x_1}^{2x_1}\left(\int_{x_1x_2}^{2x_1x_2} x_1x_2x_3\,dx_3\right)dx_2\right)dx_1$$

在 6.3 节，该积分已经计算了，结果为 179.296 9，用蒙特卡洛法求解代码如下：

```
% 构造被积函数，x 为长为 3 的列矢量或者矩阵(行数为 3)。x 的每一列表示 s 维空间中的一个点
f = @(x) prod(x);
n = 100000;
% 随机均匀生成空间中包含积分区域的长方体内的点
x1 = unifrnd(1,2,1,n);
x2 = unifrnd(1,4,1,n);
x3 = unifrnd(1,16,1,n);
ind = (x2 >= x1)&(x2 <= 2 * x1)&(x3 > x1. * x2)&(x3 < 2 * x1. * x2);
X = [x1;x2;x3];
format long
I = (4 - 1) * (16 - 1) * sum(f(X(:,ind)))/n % 用基本的蒙特卡洛法估计积分值
I =
    1.790310643580260e + 002
```

6.4.3 等分布序列的蒙特卡洛法

在区间$[a,b]$中的一个(确定性)点列$x_1,x_2,\cdots$,若对所有的有界黎曼(Riemann)可积函数$f(x)$,均有

$$\lim_{n\to+\infty}\frac{b-a}{n}\sum_{i=1}^{n}f(x_i)=\int_a^b f(x)\mathrm{d}x$$

则称该点列在$[a,b]$中是等分布的。令(ξ)表示ξ的小数部分,即$(\xi)=\xi-[\xi]$,这里$[\xi]$表示不超过ξ的最大整数。于是可以引入下面的定理:

定理 6.1 若θ为一个无理数,则数列为

$$x_n=(n\theta),\quad n=1,2,\cdots$$

在$[0,1]$中是等分布的。

定理证明从略,有兴趣读者可以参考文献[25]。

对于一般的区间$[a,b]$,可以令$u_n=x_n(b-a)+a$来得到$[a,b]$中等分布的点列。对于s重积分$I=\int_{D_s}f(p)\mathrm{d}p$,一般是挑选$s$个对有理数线性对立的无理数$\theta_1,\theta_2,\cdots,\theta_s$,来得到包含积分区域$D_s$的超长方体内的均匀分布的点列

$$P_n=((n\theta_1)(b_1-a_1)+a_1,(n\theta_2)(b_2-a_2)+a_2,\cdots,(n\theta_s)(b_s-a_s)+a_s)$$

其中,$[a_i,b_i]$是包含积分区域的超长方体的第$i(i=1,2,\cdots,s)$维边长。无理数对有理数线性独立是指对$\alpha_j\in\mathbf{Q},j=1,2,\cdots,s$不全为0,有

$$\alpha_1\theta_1+\alpha_2\theta_2+\cdots+\alpha_s\theta_s\neq 0$$

这样,可以得到用等分布序列蒙特卡洛法计算的积分的近似值:

$$I\approx M_{D_s}\overline{f}=\frac{M_C}{n}\sum_{i\in D_s}f(P_i),\quad i=1,2,\cdots,n$$

可以证明,采用等分布序列的蒙特卡洛法比采用随机序列的蒙特卡洛法误差阶要好。

下面结合例子来说明。

【例 6.4-3】 用等序列的蒙特卡洛法计算例 6.3-1 中积分:

$$\int_0^1\left(\int_0^1\left(\int_0^1\left(\int_0^1 \mathrm{e}^{x_1x_2x_3x_4}\mathrm{d}x_4\right)\mathrm{d}x_3\right)\mathrm{d}x_2\right)\mathrm{d}x_1$$

```
%构造被积函数,x为长为4的列向量或者矩阵(行数为4)。x的每一列表示四维空间中的一个点
f = @(x) exp(prod(x));
n = 10000;
%选取对有理数独立的无理数 sqrt(2),sqrt(3),sqrt(6)/3,sqrt(10)来生成等分布序列
x = bsxfun(@times,repmat(1:n,4,1),[sqrt(2);sqrt(3);sqrt(6)/3;sqrt(10)]);
x = mod(x,1);
format long
I = sum(f(x))/n %用基本的蒙特卡洛法估计积分值
I =
    1.069297245824625
```

【例 6.4-4】 用等序列的蒙特卡洛法计算例 6.3-4 中的积分,并和基本的蒙特卡洛法比较:

$$\int_0^1 \mathrm{d}x_1 \int_{\exp(x_1)/2}^{\exp(x_1)} \mathrm{d}x_2 \int_{(x_1+\sin(x_2))/2}^{x_1+\sin(x_2)} \mathrm{d}x_3 \int_{(x_1+x_3)/2}^{x_1+x_3} \mathrm{d}x_4 \rightarrow$$

$$\leftarrow \int_{x_4}^{2x_4} (\sin(x_1 \exp(x_2 \sqrt{x_3})) + x_4^{x_5})\mathrm{d}x_5$$

该五重积分的积分区域是不规则的，在6.3节，利用nIntegrate函数给出了其精度比较高的解，但是也看出，求解时间相当长。这里利用蒙特卡洛法再重新对其计算并进行结果比较。

从上述积分表达式容易找到包含被积区域的一个超长方体如下：

$$0 \leqslant x_1 \leqslant 1, \frac{1}{2} \leqslant x_2 \leqslant \mathrm{e}, \frac{\sin(\mathrm{e})}{2} \leqslant x_3 \leqslant 2, \frac{\sin(\mathrm{e})}{4} \leqslant x_4 \leqslant 3, \frac{\sin(\mathrm{e})}{4} \leqslant x_5 \leqslant 6$$

选取$\sqrt{2}$，$\sqrt{3}$，$\frac{\sqrt{6}}{3}$，$\sqrt{10}$，$\sqrt{19}$这五个对有理数独立的无理数来生成等分布序列。利用等序列的蒙特卡洛法计算，代码如下：

```
clear
format long
%构造被积函数
f = @(x) sin(x(1,:). * exp( x(2,:). * sqrt(x(3,:)) )) + x(4,:).^x(5,:);
n = 1000000;
%选取对有理数独立的无理数 sqrt(2),sqrt(3),sqrt(6)/3,sqrt(10)来生成等分布序列
x = bsxfun(@times,repmat(1:n,5,1),[sqrt(2);sqrt(3);sqrt(6)/3;sqrt(10);sqrt(19)]);
x = mod(x,1);
%进行变换,使得区间(0,1)上的等分布序列变到各层积分区间上去
BminusA = diff([0.5 sin(exp(1))/2 sin(exp(1))/4 sin(exp(1))/4;exp(1) 2 3 6])';
x(2:end,:) = bsxfun(@times,x(2:end,:),BminusA);
x(2:end,:) = bsxfun(@plus,x(2:end,:),[0.5;sin(exp(1))/4 * [2;1;1]]);
%判断哪些点落入积分区域
ind = ( x(2,:)>= exp(x(1,:))/2 )&( x(2,:)<= exp(x(1,:)) )&...
( x(3,:)>= (sin(x(2,:)) + x(1,:))/2 )&( x(3,:)<= (sin(x(2,:)) + x(1,:)))&...
( x(4,:)>= (x(1,:) + x(3,:))/2 )&( x(4,:)<= x(1,:) + x(3,:) )&...
( x(5,:)>= x(4,:) )&(x(5,:)<= 2 * x(4,:));
%求近似积分
I1 = (exp(1) - 0.5) * (2 - sin(exp(1))/2) * (3 - sin(exp(1))/4) * (6 - sin(exp(1))/4) * ...
    sum(f(x(:,ind)))/n
I1 =
    5.898580893966090
```

利用一般的蒙特卡洛法计算，代码如下：

```
clear
format long
%构造被积函数
f = @(x) sin(x(1,:). * exp( x(2,:). * sqrt(x(3,:)) )) + x(4,:).^x(5,:);
n = 1000000;
%生成超长方体内的随机数
x(1,:) = rand(1,n);
x(2,:) = unifrnd(0.5,exp(1),1,n);
x(3,:) = unifrnd(sin(exp(1))/2,2,1,n);
```

```
x(4,:) = unifrnd(sin(exp(1))/4,3,1,n);
x(5,:) = unifrnd(sin(exp(1))/4,6,1,n);
% 判断哪些点落入积分区域
ind = ( x(2,:)>= exp(x(1,:))/2 )&( x(2,:)<= exp(x(1,:)) )&...
( x(3,:)>= (sin(x(2,:)) + x(1,:))/2 )&( x(3,:)<= (sin(x(2,:)) + x(1,:)))&...
( x(4,:)>= (x(1,:) + x(3,:))/2 )&( x(4,:)<= x(1,:) + x(3,:) )&...
( x(5,:)>= x(4,:) )&(x(5,:)<= 2 * x(4,:));
% 求近似积分
I2 = (exp(1) - 0.5) * (2 - sin(exp(1))/2) * (3 - sin(exp(1))/4) * (6 - sin(exp(1))/4) * ...
    sum(f(x(:,ind)))/n
I2 =
    5.682118433926163
```

一般的蒙特卡洛法具有计算不可重复性等缺点，相比而言，等序列的蒙特卡洛法在选定产生等序列的无理数后，计算结果就唯一确定了。从上面讨论可以看出，如果对于积分精度要求不是很高，蒙特卡洛法在求解重数较多的积分时是很有效的。

6.5 案例5：第二类 Fredholm 积分方程数值求解

6.5.1 概　述

积分方程是指方程中含有积分，而积分内外都含有未知函数的方程。积分方程有各种不同的类型，不同类型方程的理论及解法也有很大差异。本案例介绍第二类 Fredholm 积分方程的数值解法。

第二类 Fredholm 积分方程具有如下形式：

$$y(x)=\lambda\int_a^b k(x,s)y(s)\mathrm{d}s+f(x),\quad a\leqslant x\leqslant b \tag{6-1}$$

其中，$k(x,s)$称为积分方程的核，$f(x)$称为自由项；λ 为参数；$\lambda,k(x,s),f(x)$均为已知；而 $y(x)$为未知函数。

求积分方程(6-1)的解 $y(x)$的数值方法就是在区间$[a,b]$的某些点 $x_i(i=1,2,\cdots,n)$上求 $y(x_i)$的近似值 y_i，使得误差$|y(x_i)-y_i|$满足精度要求。

6.5.2 具体解法

为求解方程(6-1)，通常可用数值积分的方法将方程(6-1)离散化，设数值积分公式为

$$\int_a^b f(x)\mathrm{d}x\approx\sum_{j=1}^n A_j f(x_j) \tag{6-2}$$

其中，$x_j\in[a,b]$为求积节点，A_j 为求积系数。在方程(6-1)的积分中用求积公式(6-2)近似，则得

$$y(x)\approx\lambda\sum_{j=1}^n A_j k(x,x_j)y(x_j)+f(x) \tag{6-3}$$

若在式(6-3)中令 $x=x_1,\cdots,x_n$，并记 $y_i\approx y(x_i),k_{ij}=k(x_i,x_j),f_i=f(x_i)$，由方程(6-3)得

$$y_i\approx\lambda\sum_{j=1}^n A_j k_{ij}y_j+f_i,\quad x_i(i=1,2,\cdots,n) \tag{6-4}$$

这是关于未知量 $y_1, y_2, \cdots, y_n$ 的线性方程组，若记

$$\boldsymbol{K}=\begin{bmatrix} A_1k_{11} & A_2k_{12} & \cdots & A_nk_{1n} \\ A_1k_{21} & A_2k_{22} & \cdots & A_nk_{2n} \\ \vdots & \vdots & & \vdots \\ A_1k_{n1} & A_nk_{n2} & \cdots & A_nk_{nn} \end{bmatrix}$$

则当方程(6-4)的系数行列式(称为核 $k(x,s)$ 关于求积系数 $A_j(j=1,2,\cdots,n)$ 的 Fredholm 行列式)

$$\det(\boldsymbol{I}-\lambda\boldsymbol{K})\neq 0 \tag{6-5}$$

时，方程组有唯一解 $y_1, y_2, \cdots, y_n$，它就是积分方程(6-1)的解 $y(x)$ 在点 $x_1, x_2, \cdots, x_n$ 上的近似值。由此可得到 $y(x)$ 在 $[a,b]$ 上的近似解：

$$\tilde{y}(x)\approx f(x)+\lambda\sum_{j=1}^{n}A_jk(x,x_j)y_j \tag{6-6}$$

积分方程数值解求解过程要解线性方程组，其阶数与求解公式节点数相同，为此需要知道相应的求积系数 A_j 和求积节点 x_j。这里介绍 Gauss-Lobatto(高斯-洛巴托)求积公式。这里仅介绍其公式，关于公式的推导有兴趣的读者可以参考文献[14]，或者其他相关的数值计算方面的书。Gauss-Lobatto 求积公式的表达式如下：

$$\int_{-1}^{1}f(x)\mathrm{d}x=\frac{2}{n(n+1)}[f(-1)+f(1)]+\sum_{j=2}^{n-1}A_jf(x_j)+R_n[f] \tag{6-7}$$

Gauss-Lobatto 求积公式的系数和余项分别为：

$$A_j=\frac{2}{n(n+1)[P_n(x_j)]^2} \tag{6-8}$$

$$R_n[f]=-\frac{n^3(n+1)2^{2n+1}[(n-1)!]^4}{(2n+1)[(2n)!]^3}f^{(2n)}(\eta),\quad \eta\in(-1,1) \tag{6-9}$$

其中，x_j 为 $P'_n(x)$ 的零点；$P_n(x)$ 为 n 次 Legendre(勒让德)多项式。

Legendre 多项式具有如下表达式：

$$P_0(x)=1$$

$$P_n(x)=\frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}[(x^2-1)^n] \tag{6-10}$$

$$n=1,2,\cdots,\quad x\in[-1,1]$$

Gauss-Lobatto 求积公式要求被积区间为 $[-1,1]$，对于一般被积区间 $[a,b]$，在应用 Gauss-Lobatto 求积公式时，需要作如下变量替换：

$$s=\frac{1}{2}[(b-a)t+(b+a)]$$

将区间 $[a,b]$ 映射到区间 $[-1,1]$。

至此，根据前面介绍的这些知识，可以写出求解第二类 Fredholm 积分方程的程序，代码如下：

```
function [ybar,ybarSym] = Fredholm2(ker,f,lambda,a,b,n)
% 求解第二类 Fredholm 积分方程:y(x) = lambda * int(k(x,s) * y(s) * ds,a,b) + f(x),a <= x <= b
% ker,核函数 k(x,s)的函数句柄;f,自由项 f(x)的函数句柄;lambda,参数;a,b 积分上下限
% n,Gauss - Lobatto 积分公式的积分节点数
```

```
% ybar,y(x)的近似函数,以函数句柄形式返回
% ybarSym,y(x)的近似函数,与 ybar 等价,为方便查看以符号公式形式返回
eval(['ker=',vectorize(ker),';']);                   % ker 统一为点乘形式
eval(['f=',vectorize(f),';']);                       % f 统一为点乘形式
if nargin<=5
    % 没有指定 n 的话,n 为 5
    n=5;
end
P=LegendrePoly(n);                                   % 调用 LegendrePoly 求 n 次 Legendre 多项式
diffP=diff(P);                                       % n 次 n 次 Legendre 多项式的导数
% 上面得到的 diffP 还是符号多项式表达式,利用 sym2poly 函数将其转化成一元 n 次多项式的标准
% 形式,即从最高次到常数项的所有系数组成的向量,利用 roots 求其根,并利用 sort 从小到大排
% 好序和 -1,1 一起得到 n 次 Legendre 多项式的求积节点
xk=[-1;sort(roots(sym2poly(diffP)));1];
% 计算求积系数
Ak=(2/(n^2+n)).*[1;1./(polyval(sym2poly(P),xk(2:end-1)).^2);1];
if a~=-1 || b~=1
    % 如果[a,b]不是[-1,1]的话,进行变量替换
    tmp=(b-a)/2;
    xab=tmp*xk+(a+b)/2;
    F=f(xab);                                        % 得到自由项 f(x)关于求积节点的函数值向量
    if isscalar(F)                                   % 针对某些自由项函数为常数的情形
        F=F*ones(size(xab));
    end
    [Y1,X1]=meshgrid(xab,xab);
    kerXY=ker(X1,Y1);
    if isscalar(kerXY) % 针对某些核函数为常数的情形
        kerXY=kerXY*ones(size(X1));
    end
    K=bsxfun(@times,kerXY,Ak');                      % y 的线性方程组的系数矩阵
    K=(b-a)/2*lambda*K;                              % 进行积分变量替换后的 K
    Y=(eye(n+1)-K)\F;                                % 求解线性方程组得到 Y
    X=xab;                                           % 对应的自变量 X 的值
else                                                 % 积分区间为[-1,1],不用变量替换
    F=f(xk);
    if isscalar(F)
        F=F*ones(size(xk));
    end
    [Y1,X1]=meshgrid(xk,xk);
    kerXY=ker(X1,Y1);
    if isscalar(kerXY)                               % 针对某些核函数为常数的情形
        kerXY=kerXY*ones(size(X1));
    end
    K=lambda*bsxfun(@times,kerXY,Ak');               % Y 的线性方程组的系数矩阵
    Y=(eye(n+1)-K)\F;
    X=xk;
end
fun=cell(n+1,1);
```

```
for k = 1 : n + 1
    % 积分离散化后每一个节点上的分量函数
    fun{k} = @(x) Ak(k) * Y(k) * ker(x,X(k));
end
ybar = @approxY; % 返回 y(x)的近似函数
    function yx = approxY(x)
        % 利用嵌套函数形式可以方便的计算 ybar 的近似函数
        subY = fun{1}(x);
        for jj = 2 : n + 1
            subY = subY + fun{jj}(x);
        end
        yx = f(x) + (b - a)/2 * lambda * subY;
    end
% 下面利用符号计算得到 ybar 的数学表达式,读者可以根据需要采用该方法或者采用
% 上面利用嵌套函数返回函数句柄的做法
ybarSym = GenerateExpr(f,a,b,lambda,Ak,ker,X,Y);
end
%=========================================
% 计算 n 次 Legendre 多项式的函数(普通子函数形式)
%=========================================
function P = LegendrePoly(n)
syms x;
if n == 1
    P = sym(1);
    return
else
    P = simplify( diff((x^2 - 1)^n,n)/2^n/gamma(n + 1));
end
end
%=========================================================
% 符号计算得到 ybar 的数学表达式,之所以采用普通子函数是因为
% 含有嵌套函数情况下,主函数或者嵌套函数不允许动态生成符号变量
%=========================================================
function ybarSym = GenerateExpr(f,a,b,lambda,Ak,ker,X,Y)
digits(16);
As = sym(Ak,'d');
Xs = sym(X,'d');
Ys = sym(Y,'d');
syms x;
ybarSym = f(x) + (b - a)/2 * lambda * sum(As. * ker(x,Xs). * Ys);
ybarSym = simplify(ybarSym);
end
```

该程序除了可以计算第二类 Fredholm 积分方程,其把符号计算和数值计算结合的做法也可以供读者参考。在求 n 次 Legendre 多项式以及其导数的时候,采用的是符号计算的方式,而求积分系数以及积分节点的时候分别利用 poly2sym 将符号多项式转化成数值型的,再利用针对数值多项式的函数 polyval 或者 roots 求多项式在某些点上的值或者多项式的根。

程序最后在求未知函数的近似函数时，采用了两种方法：一种是纯数值的方法，通过建立一个内嵌函数 yx=approxY(x)来表示未知函数，并返回该函数的句柄；另一种是采用符号计算的方式得到其表达式。读者在实际应用中可以酌情使用这两种方法。

6.5.3 实 例

下面看一个第二类 Fredholm 积分方程求解的例子：

【例 6.5-1】 求解积分方程

$$y(x)=\int_0^1(1-3xs)y(s)\mathrm{d}s+(1-3x)$$

其解析解为

$$y(x)=\frac{2}{3}(1-3x)$$

利用 Fredholm 2 函数求其解，代码如下：

```
format long
ker = @(x,s)1 - 3 * x * s;
f = @(x)(1 - 3 * x);
lambda = 1;n = 3;
[ybar,ybarSym ] = Fredholm2(ker,f,lambda,0,1,n)
ybar =
    @Fredholm2/approxY
ybarSym =
0.6666666666666666 - 2.0 * x
```

从上面的求解结果来看，和解析解相比，仅仅采用三个点的 Gauss - Lobatto 求积公式得到的近似解和解析解的差异就已经非常小了。ybar 是函数句柄形式，如果不方便观察其公式，可以从其等价的符号表达式 ybarSym 看出近似解几乎和解析解完全相等。可以把得到的近似解和解析解在一个图里画出来：

```
clear
ker = @(x,s)1 - 3 * x * s;
f = @(x)(1 - 3 * x);
lambda = 1;n = 3;
[ybar,ybarSym ] = Fredholm2(ker,f,lambda,0,1,n);
x = linspace(0,1,50);
AnalyticSol = @(x)2 * (1 - 3 * x)/3;
AnalyticY = AnalyticSol(x);
ybarX = ybar(x); % 利用返回的 y(x)近似解的函数句柄 ybar 得到对应的 y(x);
eval(['ybar2 = @(x)',char(ybarSym),';']) % 将近似解的符号表达式转化成另一个函数句柄 ybar2
ybarSymx = ybar2(x);
plot(x,AnalyticY,'*')
hold on
plot(x,ybarX,'o')
plot(x,ybarSymx,'d');hold off;
legend('AnalyticY','ybarX','ybarSymx','location','best')
```

得到的图形如图 6.2 所示。

AnalyticY—解析解；YbarX—数值解；YbarSymx—与数值解对应的表达式形式

图 6.2　数值解与解析解对比

从图 6.2 可以看出，解析解和数值解几乎完全重合。再看一个例子：

【例 6.5-2】 求解积分方程：

$$y(x)=\frac{2}{e-1}\int_0^1 e^x y(s)\,ds-e^x$$

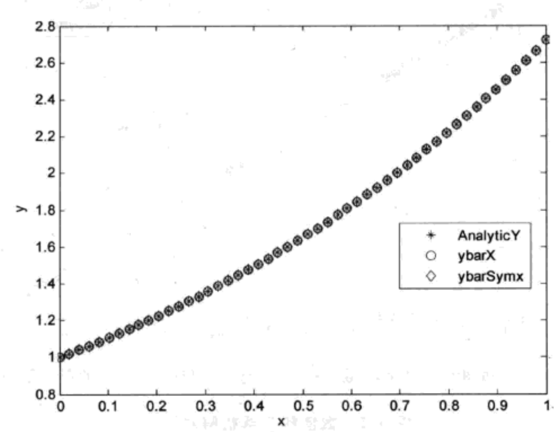
其解析解为

$$y(x)=e^x$$

利用 Fredholm2 函数求其数值解，代码如下：

```
>> clear
format long
ker = @(x,s)exp(x);
f = @(x) - exp(x);
lambda = 2/(exp(1) - 1);n = 5;
[ybar,ybarSym ] = Fredholm2(ker,f,lambda,0,1,n)
ybar =
    @Fredholm2/approxY
ybarSym =
0.9999999999990871 * exp(x)
>> x = linspace(0,1,50);
AnalyticY = exp(x);
ybarX = ybar(x); % 利用返回的 y(x)近似解的函数句柄 ybar 得到对应的 y(x);
eval(['ybar2 = @(x)',char(ybarSym),';']) % 将近似解的符号表达式转化成另一个函数句柄 ybar2
ybarSymx = ybar2(x);
plot(x,AnalyticY,'*')
hold on
plot(x,ybarX,'o')
plot(x,ybarSymx,'d');hold off;
legend('AnalyticY','ybarX','ybarSymx','location','best')
```

从上面得到的近似解的表达式 ybarSym=0.999 999 999 999 087 1 * exp(x)可以看出，仅采用五点的 Gauss - Lobatto 求积公式，求出的数值解的表达式和解析解的表达式相差就不到千亿分之一了。解析解和数值解在一张图中已经无法分辨出区别，如图 6.3 所示。



AnalyticY—解析解；YbarX—数值解；YbarSymx—与数值解对应的表达式形式

图 6.3 数值解与解析解对比

由此可见，上面介绍的数值求解积分方程的方法是非常有效的一种方法。

6.6 案例6：第一类 Fredholm 积分方程数值求解

6.6.1 概　述

许多实际问题的模型可建立第一类 Fredholm 积分方程，例如，医用 X 射线断层摄影技术问题、遥感勘测数据问题、地震勘测问题等建立的数学模型通常都是第一类 Fredholm 积分方程，即

$$\int_a^b k(x,s)y(s)\mathrm{d}s = f(x), \quad a \leqslant x \leqslant b \tag{6-11}$$

其中，$k(x,s)$是已知的实的平方可积函数；自由项 $f(x)$已知；$y(s)$是未知函数。

虽然第一类 Fredholm 积分方程形式比第二类 Fredholm 积分方程简单，但其求解相对第二类 Fredholm 积分方程要困难许多。例如，由方程(6-11)左边可知，函数 $f(x)$一定是连续的，否则方程两端不可能相等，因而没有可积的解。因此，第一类 Fredholm 积分方程的解的存在性需要进一步研究。另一方面，一些第一类 Fredholm 积分方程的解不止一个，譬如方程

$$\int_0^\pi x \cdot \sin s \cdot y(s)\mathrm{d}s = x, \quad 0 \leqslant x \leqslant \pi$$

有解

$$y(s) = \frac{1}{2}$$

但是,可以验证具有如下表达式的函数都是上述方程的解：

$$y(s)=\frac{1}{2}+\sin nx,\quad n=1,2,\cdots$$

因此,方程解的唯一性也需要探讨。事实上,当前仅对一些含有特殊类型核的第一类 Fredholm 积分方程有比较有效的解法,譬如退化核、对称核等。好多求解方法的求解前提都建立在第一类 Fredholm 方程满足比较强的条件基础上。

本案例只讨论一类可以转化为第二类 Fredholm 积分方程的第一类 Fredholm 积分方程求解方法,并简单讨论了直接数值求解的方法。

6.6.2 一类可以化为第二类 Fredholm 积分方程的第一类 Fredholm 积分方程求解方法

现在考虑第一类 Fredholm 积分方程(6-11),如果核 $k(x,s)$在任一与 s 轴平行的线只相交一次的对角曲线 $s=\xi(x)$上有非零有限跳跃,即

$$k(x,\xi(x)-0)-k(x,\xi(x)+0)=L(x)$$

其中,"−0"表示左趋近,类似的,"+0"表示右趋近。则(6-11)可以写为

$$\int_a^{\xi(x)}k(x,s)y(s)\mathrm{d}s+\int_{\xi(x)}^b k(x,s)y(s)\mathrm{d}s=f(x),\quad a\leqslant x\leqslant b \tag{6-12}$$

方程(6-12)两边对变量 x 求导,得

$$\begin{aligned}&\int_a^{\xi(x)}\frac{\partial k}{\partial x}y(s)\mathrm{d}s+k(x,\xi(x)-0)y(\xi(x))\xi'(x)-k(x,a)y(a)a'+\\&\int_{\xi(x)}^b\frac{\partial k}{\partial x}y(s)\mathrm{d}s+k(x,b)y(b)b'-k(x,\xi(x)+0)y(\xi(x))\xi'(x)=\\&\int_a^b\frac{\partial k}{\partial x}y(s)\mathrm{d}s+L(x)y(\xi(x))\xi'(x)=f'(x)\end{aligned} \tag{6-13}$$

式(6-13)还不是标准的第二类 Fredholm 积分方程形式,需要把 $\xi(x)$用另一个变量,譬如 τ 替换掉,因 $\tau=\xi(x)$,进行逆变换得到 $x=\eta(x)$,代入式(6-13),由于假设 $L(x)$非零,因此整理得到

$$y(\tau)+\int_a^b\frac{\left.\dfrac{\partial k(x,s)}{\partial x}\right|_{x=\eta(\tau)}}{L(\eta(\tau))\xi'(\eta(\tau))}y(s)\mathrm{d}s=\frac{f'(\eta(\tau))}{L(\eta(\tau))\xi'(\eta(\tau))} \tag{6-14}$$

记

$$K(\tau,s)=\frac{-\left.\dfrac{\partial k(x,s)}{\partial x}\right|_{x=\eta(\tau)}}{L(\eta(\tau))\xi'(\eta(\tau))},$$

$$F(\tau)=\frac{f'(\eta(\tau))}{L(\eta(\tau))\xi'(\eta(\tau))}$$

将其带入(6-14)式,得到

$$y(\tau)=\int_a^b K(\tau,s)y(s)\mathrm{d}s+F(\tau) \tag{6-15}$$

于是就得到了标准的第二类 Fredholm 积分方程表达式。

细心的读者可能已经发现,式(6-12)右端项 $f(x)$如果加上任意常数 C,按照上述方法也能得到(6-15)式,因此式(6-15)的解不一定是原方程(6-11)的解。事实上,有下面定理：

定理 6.2

1) 核 $k(x,s)$ 在 $x-s$ 轴上有界，且除曲线 $s=\xi(x)$ 外处处连续；

2) $\xi(x)$ 是连续的且其导函数是不为零的连续函数，并有

$$\xi(a)=a,\quad \xi(b)=b;$$

3) $k(x,\xi(x)-0)$，$k(x,\xi(x)+0)$ 在 (a,b) 上存在且它们的差：

$$L(x)=k(x,\xi(x)-0)-k(x,\xi(x)+0)$$

是连续的；

4) $k(x,s)$ 对变量 x 可导且 $\frac{\partial k}{\partial x}$ 满足 1) 的性质，且当 $s<\xi(x)$ 以及 $s>\xi(x)$ 时，$\frac{\partial k}{\partial x}$ 表达式一样；

5) $f(x)$，$f'(x)$ 是连续的。

在这些条件下，如果核

$$\frac{-\left.\frac{\partial k(x,s)}{\partial x}\right|_{x=\eta(\tau)}}{L(\eta(\tau))\xi'(\eta(\tau))}$$

的 Fredholm 行列式不为零，则第一类 Fredholm 积分方程

$$\int_a^b k(x,s)y(s)\mathrm{d}s = f(x)+C,\quad a\leqslant x\leqslant b$$

仅对于参数 C 的某一特定值有唯一连续解。这个解就是第二类 Fredholm 积分方程 (6-15) 的唯一连续解。当 $L(x)$ 总为零，如果

$$M(x)=\frac{\partial k(x,\xi(x)-0)}{\partial x}-\frac{\partial k(x,\xi(x)+0)}{\partial x}$$

不等于 0，上述定理可以推广。

当第一类 Fredholm 积分方程满足定理 6.2 的条件时，可以利用下面程序转化成第二类 Fredholm 积分方程，再求其数值解的代码如下：

```
function [ybar,ybarSym] = Fredholm1to2(ker1,ker2,f,xi,invxi,a,b,n)
% 求解满足定理 6.2 条件的第一类 Fredholm 积分方程。方法是转化成第二类 Fredholm 积分方程再求解
% ker1 函数句柄，当 s 小于 xi(x)时，原积分方程的核的表达式；
% ker2 函数句柄，当 s 大于 xi(x)时，原积分方程的核的表达式；
% f，方程右端自由项 f(x)的函数句柄；
% xi，核函数的定义域上的不连续曲线，s 关于 x 的函数，也是以函数句柄形式给出
% invxi，xi 的反函数，也是以函数句柄形式给出
% a,b 积分上下限，n，再利用 Fredholm2 函数求解时 Gauss - Lobatto 积分公式的积分节点数
%  ybarSym，y(x)的近似函数，与 ybar 等价，为方便查看以符号公式形式返回
syms x s tao
k1 = ker1(x,s); % 得到符号表达式，为了方便公式推导，为转化成第二类 Fredholm 积分方程做准备
k2 = ker2(x,s);
fx = f(x);
xix = xi(x);
invxit = invxi(tao);
diffkx = diff(k1,x); % 核关于 x 的偏导数，diff(k2,x)应等于 diff(k1,x)
Lx = k1 - k2;
diffxix = diff(xix,x);
```

```
difffx = diff(fx,x);
Kxt = subs( - diffkx/(Lx * diffxix),x,invxit); % 得到变换后的第二类 Fredholm 积分方程的核
Ft = subs(difffx/(Lx * diffxix),x,invxit); % 得到变换后的第二类 Fredholm 积分方程的自由项
eval(['K = @(tao,s)' char(Kxt),';']);
eval(['F = @(tao)' char(Ft),';']);
[ybar,ybarSym] = Fredholm2(K,F,1,a,b,n);
```

【例 6.6-1】 求解第一类 Fredholm 积分方程：

$$\int_0^1 K(x,s)y(s)\mathrm{d}s = 67x^3 - 40x^2 + 52x + C$$

并确定常数 C。其中，$K(x,s)=\begin{cases} xs(x-s)^2+1, & s<x \\ xs(x-s)^2, & s>x \end{cases}$

该方程的解析解为 $y(x)=60x^2+60x+24$，常数 $C=0$。

容易验证此方程满足定理 6.2 的条件，利用 Fredholm1to2 函数求解代码如下：

```
>> ker1 = @(x,s)x * s * (x - s)^2 + 1;
ker2 = @(x,s)x * s * (x - s)^2;
f = @(x) 67 * x^3 - 40 * x^2 + 52 * x + rand;
xi = @(x)x;
invxi = @(x)x;
a = 0;
b = 1;
n = 5;
[ybar,ybarSym] = Fredholm1to2(ker1,ker2,f,xi,invxi,a,b,n)
ybar =
    @Fredholm2/approxY
ybarSym =
59.9999999999999 * x^2 + 60.00000000000017 * x + 23.99999999999995
```

可见求出来的近似解已经非常接近解析解了。将 ybarSym 表达式代入原方程，可以确定常数 $C=0$，即

```
>> syms x t
y1 = int((x * t * (x - t)^2 + 1) * (59.9999999999999 * t^2 + 60.00000000000017 * t + ...
23.99999999999995),t,0,x);
y2 = int((x * t * (x - t)^2) * (59.9999999999999 * t^2 + 60.00000000000017 * t + ...
23.99999999999995),t,x,1);
C = simplify(y1 + y2 - 67 * x^3 + 40 * x^2 - 52 * x)
C =
0
```

【例 6.6-2】 求解第一类 Fredholm 积分方程：

$$\int_0^1 K(x,s)y(s)\mathrm{d}s = \frac{2}{3}x^3 + \frac{5}{2}x^2 - 3x + C$$

并确定常数 C。其中，$K(x,s)=\begin{cases} x^2+x-2s+1, & s<\frac{1}{2}(x^2+x) \\ x^2+x-2s, & s>\frac{1}{2}(x^2+x) \end{cases}$

此时 $\xi(x)=\frac{1}{2}(x^2+x)$，其反函数有两个，根据 $x>0$，淘汰其中一个得到其反函数为

$$\eta(x)=\sqrt{2x+\frac{1}{4}}-\frac{1}{2}$$

利用 Fredholm1to2 函数求解代码如下：

```
ker1 = @(x,s)x^2 + x - 2 * s + 1;
ker2 = @(x,s)x^2 + x - 2 * s;
f = @(x) 2 * x^3/3 + 2.5 * x^2 - 3 * x;
xi = @(x)(x^2 + x)/2;
invxi = @(x)(2 * x + 1/4)^(1/2) - 1/2;
a = 0;
b = 1;
n = 10;
[ybar,ybarSym] = Fredholm1to2(ker1,ker2,f,xi,invxi,a,b,n);ybarSym
ybarSym =
(4.0 * x)/(2.0 * x + 0.25)^(1/2) - 4.5/(2.0 * x + 0.25)^(1/2) + 2.888889510215005
```

将 ybarSym 代入原方程，求 C，代码如下：

```
syms x t
y1 = int((x^2 + x - 2 * t + 1) * ((4.0 * t)/(2.0 * t + 0.25)^(1/2) - 4.5/(2.0 * t + 0.25)^(1/2)... +
2.888889510215005),t,0,(x^2 + x)/2);
y2 = int((x^2 + x - 2 * t) * ((4.0 * t)/(2.0 * t + 0.25)^(1/2) - 4.5/(2.0 * t + 0.25)^(1/2)...
 + 2.888889510215005),t,(x^2 + x)/2,1);
C = simplify(y1 + y2 - 2 * x^3/3 - 2.5 * x^2 + 3 * x);
eval(['Cx = @(x)' vectorize(C),';']);
C0 = Cx(0 : 0.1 : 1)
mean(C0)
C0 =
  Columns 1 through 10
   -1.2056   -1.2056   -1.2056   -1.2056   -1.2056   -1.2056   -1.2056   -1.2056
   -1.2056   -1.2056
  Column 11
   -1.2056
ans =
   -1.2056
```

求得常数 C 约为 -1.2056。

6.6.3 第一类 Fredholm 积分方程的直接数值积分解法讨论

上节给出了一种将第一类 Fredholm 积分方程转化为第二类 Fredholm 积分方程再数值求解的方法。读者自然会想到直接数值积分的方法。事实上，直接数值积分的方法在很多时候是行不通的，这是因为按照类似 6.5 节的方法，取一种数值积分公式

$$\int_a^b g(x)\mathrm{d}x \approx \sum_{j=1}^{n} A_j g(x_j) \tag{6-16}$$

其中，$x_1,x_2,\cdots,x_n$ 为区间$[a,b]$中求积节点，$A_1,A_2,\cdots,A_n$ 为求积系数(求积权重)。离散化方程(6-11)，得到

$$\sum_{j=1}^{n} A_j k_{ij} y_j = f_i, \quad i=1,2,\cdots,n \tag{6-17}$$

其中，$k_{ij}=k(x_i,x_j)$；$y_j=y(x_j)$；$f_i=f(x_i)$。

方程(6-17)的系数矩阵在很多时候都是“病态”的，或者非满秩的，这样也就无法求出其数值解。对于一些“病态”矩阵，即使采用符号计算的方法，得到其对应的 y_j。y_j 也是关于求积节点有意义，而并不能通过插值或者拟合得到区间$[a,b]$上的整个 $y(x)$。有兴趣的读者可以参照6.5节的方法编写程序验证。

6.7 案例7：第二类 Volterra 积分方程数值求解

6.7.1 概述

第二类 Volterra 积分方程具有如下表达式：

$$y(x)=\lambda\int_a^x k(x,s)y(s)\mathrm{d}s+f(x), \quad a\leqslant s\leqslant x\leqslant b \tag{6-18}$$

其中，$k(x,s)$为积分方程的核，定义在 $a\leqslant s\leqslant x\leqslant b$ 上；$f(x)$为自由项，定义在 $a\leqslant x\leqslant b$ 上；λ 为参数。$\lambda,k(x,s),f(x)$均为已知，而 $y(x)$为未知函数。

从式(6-18)可以看出，第二类 Volterra 积分方程和第二类 Fredholm 积分方程的区别在于积分限，第二类 Volterra 积分方程的积分上限是变量，而第二类 Fredholm 积分方程上限是常量。

一般情况下，求解第二类 Fredholm 积分方程通常选择 Gauss 求积公式，而 Newton-Cotes 积分公式更适合第二类 Volterra 积分方程的数值求解。

6.7.2 具体解法

当 $a\leqslant x<s\leqslant b$ 时，定义 $k(x,s)=0$，则 Volterra 积分方程(6-18)可以看成是第二类 Fredholm 积分方程。类似6.5节，在方程(6-18)中，令 $\boldsymbol{X}=[x_0,x_1,x_2,\cdots,x_n]^{\mathrm{T}}$，其中 $x_0=a$。然后采用某种数值求积公式，譬如 Newton-Cotes 积分公式，离散化方程(6-18)，得到下式：

$$y_i \approx \lambda\sum_{j=0}^{i} A_j^i k_{ij} y_j + f_i, \quad i=0,1,2,\cdots,n \tag{6-19}$$

式(6-19)中，$y_i=y(x_i)$，$k_{ij}=k(x_i,x_j)$，$f_i=f(x_i)$ $(i=0,1,2,\cdots,n)$，A_j^i $(i\geqslant1,j=0,1,\cdots,i)$是 Newton-Cotes 求积公式的系数，将在稍后介绍，为了表达式统一，记 $A_0^0=0$。

把方程组(6-19)写成如下的矩阵形式：

$$\boldsymbol{Y}=\boldsymbol{KY}+\boldsymbol{F} \tag{6-20}$$

其中，

$$\boldsymbol{Y}=[y_0,y_1,\cdots,y_n]^{\mathrm{T}}$$

$$
\boldsymbol{K}=\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ \lambda A_0^1 k_{10} & \lambda A_1^1 k_{11} & 0 & 0 & \cdots & 0 \\ \lambda A_0^2 k_{20} & \lambda A_1^2 k_{21} & \lambda A_2^2 k_{22} & 0 & \cdots & 0 \\ \lambda A_0^3 k_{30} & \lambda A_1^3 k_{31} & \lambda A_2^3 k_{32} & \lambda A_3^3 k_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & 0 \\ \lambda A_0^n k_{n0} & \lambda A_1^n k_{n1} & \lambda A_2^n k_{n2} & \lambda A_3^n k_{n3} & \cdots & \lambda A_n^n k_{nn} \end{bmatrix}
$$

$$
\boldsymbol{F}=[f_0, f_1, \cdots, f_n]^{\mathrm{T}}
$$

由此可知，$\boldsymbol{I}_{(n+1)\times(n+1)}-\boldsymbol{K}$ 是一个下三角矩阵，方程(6-20)非常方便求解。

这里要注意的是，当 $\boldsymbol{Y}$ 求出后，并不能像第二类 Fredholm 积分方程求解那样得到 $y(x)$ 的近似解

$$
\tilde{y}(x)=\lambda\sum_{j=0}^{i} A_j^i k(x, x_j) y_j + f(x) \tag{6-21}
$$

这是因为根据 Newton－Cotes 求积公式特点，式(6-21)中的 x_j 是 $[a,x]$ 中的等分节点，也就是说对于不同的 x，x_j，y_j 是要变化的。而求出的 $\boldsymbol{Y}$ 只是当 $x=b$ 时，也就是区间 $[a,b]$ 上的等分节点上的函数值，因此对于任意 $x\in[a,b]$，不能利用 $\boldsymbol{Y}$ 来离散式(6-18)得到式(6-21)。虽然在这里，$\tilde{y}(x)$ 不能写成统一表达式，但是根据得到的 $\boldsymbol{Y}$ 和 $\boldsymbol{X}$，可以采用插值或者拟合的方法得到区间其他点上的函数值。

下面介绍 Newton－Cotes（牛顿-柯特斯）求积公式。这里仅介绍其公式，关于其推导，有兴趣的读者可以参考数值计算方面的参考书。设节点 $x_i=a+ih(i=0,1,2,\cdots,n)$，$h=\dfrac{b-a}{n}$，Newton－Cotes 求积公式如下：

$$
\int_a^b g(x)\mathrm{d}x \approx (b-a)\sum_{k=0}^{n} C_k^n g(a+kh) \tag{6-22}
$$

其中

$$
C_k^n=\frac{(-1)^{n-k}}{k!(n-k)!n}\int_0^n t(t-1)\cdots(t-(k-1))(t-(k+1))\cdots(t-n)\mathrm{d}t, \quad k=0,1,2,\cdots,n
$$

C_k^n 称为柯特斯系数。Newton－Cotes 求积公式的余项为

$$
R_n[f]=\int_a^b \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x)\mathrm{d}x, \quad \xi\in[a,b]
$$

特别地，当 n 为偶数时

$$
R_n[f]=\frac{f^{(n+2)}(\eta)}{(n+2)!}\int_a^b x\omega(x)\mathrm{d}x, \quad \eta\in[a,b]
$$

其中，$\omega(x)=(x-x_0)(x-x_1)\cdots(x-x_n)$。

根据上面介绍的知识，写出利用 Newton－Cotes 公式计算第二类 Volterra 积分方程的程序代码如下：

```
function [X,Y] = Volterra2(ker,f,lambda,a,b,n)
% 利用 Newton - Cotes 积分公式求解第二类 Volterra 积分方程：
% y(x) = lambda * int(k(x,s) * y(s) * ds,a,x) + f(x),a <= s <= x <= b
% ker,核函数 k(x,s)的函数句柄；f,自由项 f(x)的函数句柄；lambda,参数；a 积分下限,[a,b]
```

```
% x 的取值范围
% n,求积分的时候,区间[a,b]被等分的个数。
% X,[a,b]上的节点
% Y,与 X 对应的 y(x)的近似值
eval(['ker=',vectorize(ker),';']);          % ker 统一为点乘形式
eval(['f=',vectorize(f),';']);              % f 统一为点乘形式
if nargin<=5
    % 没有指定 n 的话,n 为 5
    n=5;
end
A=cell(n,1); % 存储 Cotes 系数
for k=1:n
    A{k}=NewtonCotes(k);
end
K=zeros(n+1);
h=(b-a)/n;
xk=(a:h:b)';
for ix=2:n+1
    for jx=1:ix
        K(ix,jx)=(ix-1)*h*A{ix-1}(jx)*ker(xk(ix),xk(jx)); % 计算 K 矩阵
    end
end
K=lambda*K;
F=f(xk);                                    % 得到自由项 f(x)关于求积节点的函数值向量
if isscalar(F)                              % 针对某些自由项函数为常数的情形
    F=F*ones(size(xk));
end
Y=((eye(n+1)-K)\F)'; % 求解线性方程组得到 Y
X=xk'; % 对应的自变量 X 值
end
%=========================================
% 求 Cotes 系数的函数(普通子函数形式)
%=========================================
function A=NewtonCotes(n)
% n 等分求积区间时,返回 Cotes 系数
syms t;
A=zeros(n+1,1);
for k=0:n
    % 利用符号计算得到 Cotes 系数中被积函数的表达式
    symfun=prod(t-[0:k-1,k+1:n]);
    0 % 将得到的被积函数表达式转成"点乘"形式并生成相应的函数句柄
    eval(['fun=@(t)',vectorize(symfun),';']);
    A(k+1)=quadgk(fun,0,n)*(-1)^(n-k)/gamma(k+1)/gamma(n-k+1)/n;
end
end
```

在本节开始提到当 $a\leqslant x<s\leqslant b$ 时,定义 $k(x,s)=0$,则 Volterra 积分方程(6-18)可以看成是第二类 Fredholm 积分方程,因此可以利用 Fredholm2 函数求解,只不过这时 ker 是分段

函数，需要注意其表示形式，这时近似函数 $\bar{y}(x)$ 的符号公式 ybarSym 就不方便生成了，因为 MATLAB 符号计算不支持符号大小比较，因此修改 Fredholm2 如下：

```
function ybar = NewFredholm2(ker,f,lambda,a,b,n)
%求解第二类 Fredholm 积分方程：y(x) = lambda * int(k(x,s) * y(s) * ds,a,b) + f(x),a<=x<=b
%该函数可以求解核函数 ker(x,s)是分段函数的情形
%ker,核函数 k(x,s)的函数句柄；分段函数的时候以逻辑形式表示：譬如：
%k(x,s) = x * s,(s<=x 时),k(x,s) = x * s + 1,(s>x)时,则
%ker = @(x,s) x * s * (s<=x) + (x * s + 1) * (s>x)
%f,自由项 f(x)的函数句柄；lambda,参数；a,b 积分上下限
%n,Gauss - Lobatto 积分公式的积分节点数
% ybar,y(x)的近似函数,以函数句柄形式返回
eval(['ker=',vectorize(ker),';']); %ker 统一为点乘形式
eval(['f=',vectorize(f),';']); %f 统一为点乘形式
if nargin<=5
    %没有指定 n 的话,n 为 5
    n=5;
end
P=LegendrePoly(n); %调用 LegendrePoly 求 n 次 Legendre 多项式
diffP=diff(P); %n 次 n 次 Legendre 多项式的导数
%上面得到的 diffP 还是符号多项式表达式,利用 sym2poly 函数将其转化成一元 n 次多项式的标准
%形式,即从最高次到常数项的所有系数组成的向量,利用 roots 求其根,并利用 sort 从小到大排
%好序和 -1,1 一起得到 n 次 Legendre 多项式的求积节点
xk=[-1;sort(roots(sym2poly(diffP)));1];
%计算求积系数
Ak=(2/(n^2+n)).*[1;1./(polyval(sym2poly(P),xk(2:end-1)).^2);1];
if a~=-1 || b~=1
    %如果[a,b]不是[-1,1]的话,进行变量替换
    tmp=(b-a)/2;
    xab=tmp*xk+(a+b)/2;
    F=f(xab); %得到自由项 f(x)关于求积节点的函数值向量
    if isscalar(F) %针对某些自由项函数为常数的情形
        F=F*ones(size(xab));
    end
    [Y1,X1]=meshgrid(xab,xab);
    kerXY=ker(X1,Y1);
    if isscalar(kerXY) %针对某些核函数为常数的情形
        kerXY=kerXY*ones(size(X1));
    end
    K=bsxfun(@times,kerXY,Ak'); %y 的线性方程组的系数矩阵
    K=(b-a)/2*lambda*K; %进行积分变量替换后的 K
    Y=(eye(n+1)-K)\F; %求解线性方程组得到 Y
    X=xab; %对应的自变量 X 的值
else %积分区间为[-1,1],不用变量替换
    F=f(xk);
```

```
    if isscalar(F)
        F = F * ones(size(xk));
    end
    [Y1,X1] = meshgrid(xk,xk);
    kerXY = ker(X1,Y1);
    if isscalar(kerXY) % 针对某些核函数为常数的情形
        kerXY = kerXY * ones(size(X1));
    end
    K = lambda * bsxfun(@times,kerXY,Ak'); % Y 的线性方程组的系数矩阵
    Y = (eye(n + 1) - K)\F;
    X = xk;
end
fun = cell(n + 1,1);
for k = 1:n + 1
    % 积分离散化后每一个节点上的分量函数
    fun{k} = @(x) Ak(k) * Y(k) * ker(x,X(k));
end
ybar = @approxY; % 返回 y(x)的近似函数
    function yx = approxY(x)
        % 利用嵌套函数形式可以方便的计算 ybar 的近似函数
        subY = fun{1}(x);
        for jj = 2:n + 1
            subY = subY + fun{jj}(x);
        end
        yx = f(x) + (b - a)/2 * lambda * subY;
    end
end
% ==========================================
% 计算 n 次 Legendre 多项式的函数(普通子函数形式)
% ==========================================
function P = LegendrePoly(n)
syms x;
if n == 1
    P = sym(1);
    return
else
    P = simplify( diff((x^2 - 1)^n,n)/2^n/gamma(n + 1));
end
end
```


6.7.3 实　例

下面看一个第二类 Fredholm 积分方程求解的例子：

【例 6.7-1】 求解积分方程：

$$y(x)=\int_0^x (s-x)y(s)\mathrm{d}s+x,\quad 0\leqslant x\leqslant 1$$

其解析解为

$$y(x)=\sin x$$

利用 Volterra2 函数求其解，过程如下：

```
clear
ker = @(x,s) s - x;
f = @(x) x;
[X,Y] = Volterra2(ker,f,1,0,1,10) % 取 10 点 Newton - Cotes 公式
X =
    0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000
Y =
    0 0.1000 0.1987 0.2955 0.3894 0.4794 0.5646 0.6442 0.7173 0.7833 0.8414
```

再利用 NewFredholm2 函数求解，注意 ker 的表示形式：

```
>> ker = @(x,s) (s - x) * (s <= x);
f = @(x) x;
ybar = NewFredholm2(ker,f,1,0,1,10) % 取 10 点 Gauss - Lobatto 公式
ybar(X)
ybar =
@NewFredholm2/approxY
ans =
    0 0.0999 0.1988 0.2954 0.3891 0.4803 0.5643 0.6437 0.7175 0.7835 0.8412
```

将两种方法得到的结果和解析解进行画图对比，如图 6.4 所示。

```
>> plot(X,sin(X),'*')
hold on
plot(X,Y,'o')
plot(X,ybar(X),'d');hold off;
legend('解析解','Volterra2 函数求解结果','NewFredholm2 函数求解结果','location','best')
```

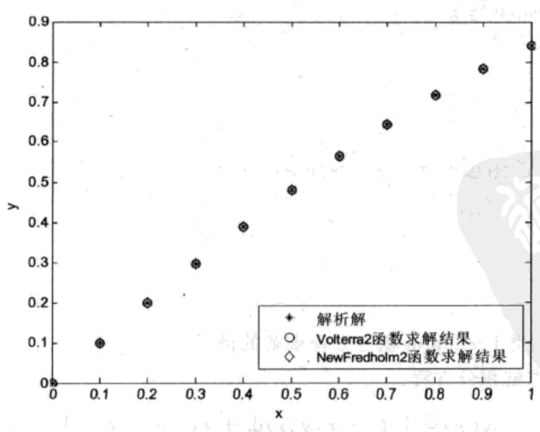


图 6.4　数值解与解析解对比

从图 6.4 可以看出，两种方法得到的数值解和解析解相差都很小，数值解几乎和解析解重合。再看一个例子：

【例 6.7-2】 求解第二类 Volterra 积分方程

$$y(x)=\int_0^x e^{-x-s}y(s)\mathrm{d}s+\frac{1}{2}(e^{-x}+e^{-3x}),\quad 0\leqslant x\leqslant 1$$

其解析解为

$$y(x)=e^{-x}$$

分别利用 Volterra2 函数以及 NewFredholm2 函数求其解，并画数值解和解析解的对比图，如图 6.5 所示。

```
>> clear
ker = @(x,s) exp(-x-s);
f = @(x) (exp(-x) + exp(-3*x))/2;
[X,Y] = Volterra2(ker,f,1,0,1,10); % 取 10 点 Newton-Cotes 公式
ker = @(x,s) exp(-x-s) * (s<=x);
ybar = NewFredholm2(ker,f,1,0,1,10); % 取 10 点 Gauss-Lobatto 公式
plot(X,exp(-X),'*')
hold on
plot(X,Y,'o')
plot(X,ybar(X),'d');hold off;
legend('解析解','Volterra2 函数求解结果','NewFredholm2 函数求解结果','location','best')
```

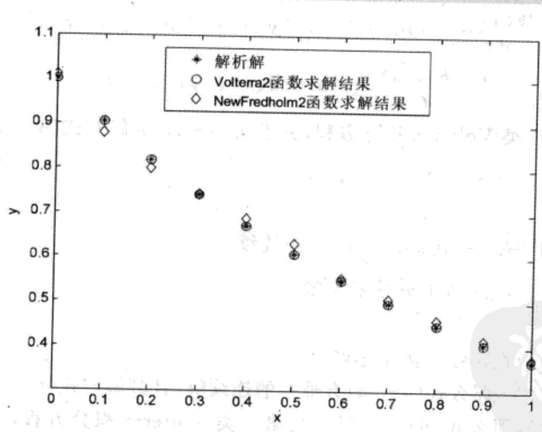


图 6.5　数值解与解析解对比

两种方法的数值解和解析解误差都不大，尤其是采用 Newton-Cotes 公式的 Volterra2 函数得到的数值解几乎和解析解重合。

6.8 案例8：第一类 Volterra 积分方程数值求解

6.8.1 概 述

就像第一类 Fredholm 积分方程和第二类 Fredholm 积分方程的差别一样，第一类Volterra 积分方程和第二类 Volterra 积分方程的差别就在于第一类 Volterra 积分方程的未知函数仅出现在积分号内，即

$$\int_a^x k(x,s)y(s)\mathrm{d}s = f(x), \quad a \leqslant x \leqslant b \tag{6-23}$$

一般来说，在实际应用中，第一类 Volterra 积分方程可分为两种类型：一类是核对各变量连续可微；另一类是当 $s=x$ 时，核无界，譬如 Abel 方程就是这种类型。一般情况下，核无界的第一类积分方程的研究相当困难，目前研究还不充分，只对一些特殊的方程可以求出解析解。这里作为数值求解案例，仅讨论连续核的 Volterra 积分方程的求解方法。

由于按照类似 6.7 节的方法直接离散式(6-23)会得到不满秩的系数矩阵，因此常采用转化成第二类 Volterra 积分方程再数值求解的方法。

6.8.2 转化为第二类 Volterra 积分方程

如果要求方程的核和解是有界的，那么 $f(0)=0$，否则式(6-23)的解不存在。$f(0)=0$ 通常被称为相容性条件。一般情况下，对式(6-23)的两端关于 x 求导得

$$\int_a^x \frac{\partial k(x,s)}{\partial x}y(s)\mathrm{d}s + k(x,x)y(x)x' - k(x,a)y(a)a' =$$
$$\int_a^x \frac{\partial k(x,s)}{\partial x}y(s)\mathrm{d}s + k(x,x)y(x) = f'(x) \tag{6-24}$$

式(6-24)是第二类 Volterra 积分方程，关于式(6-24)的解和式(6-23)的解之间的关系有如下定理：

定理 6.3

1) 核 $k(x,s)$ 和 $\frac{\partial k(x,s)}{\partial x}$ 在 $a\leqslant s\leqslant x\leqslant b$ 上连续；

2) $k(x,s)$ 在 $a\leqslant s\leqslant x\leqslant b$ 上处处不为零；

3) $f(0)=0$；

4) $f(x)$ 和 $f'(x)$ 在 $a\leqslant x\leqslant b$ 上连续。

如果上述条件成立，那么式(6-23)有唯一的连续解，且其解与式(6-24)等价。

如果 $k(x,x)=0$，那么式(6-24)仍然是第一类 Volterra 积分方程，可以再对两边求偏导。因此上述定理可以推广，即

定理 6.4

对于第一类 Volterra 积分方程(6-23)，如果核 $k(x,s)$ 连续，且关于 x 有 $n+1$ 阶导数，并且

$$k(x,x)=\left[\frac{\partial k}{\partial x}\right]_{s=x}=\left[\frac{\partial^2 k}{\partial x^2}\right]_{s=x}=\cdots=\left[\frac{\partial^{n-1} k}{\partial x^{n-1}}\right]_{s=x}=0$$

而$\left[\frac{\partial^n k}{\partial x^n}\right]_{s=x}\neq 0$，如果$f(x),f'(x),\cdots,\frac{d^n}{dx^n}f(x)\bigg|_{x=0}=0$，则原方程(6-23)存在唯一的连续解，并与第二类 Volterra 积分方程(6-25)等价。

$$y(x)+\int_0^x \frac{\frac{\partial^{n+1}k(x,s)}{\partial x^{n+1}}}{\left[\frac{\partial^n k}{\partial x^n}\right]_{s=x}} y(s)\,ds=\frac{\frac{d^{n+1}}{dx^n}f(x)}{\left[\frac{\partial^n k}{\partial x^n}\right]_{s=x}} \qquad (6-25)$$

方便起见，假设当第一类 Volterra 积分方程满足定理 6.3 的条件时，可以利用下面程序转化成第二类 Volterra 积分方程，再求其数值解，代码如下：

```
function [X,Y] = Volterra1to2(ker,f,lambda,a,b,n)
% 求解满足定理 6.3 条件的第一类 Volterra 积分方程。方法是转化成第二类 Volterra 积分方程再求解
% ker 函数句柄，原积分方程的核的表达式；
% f，方程右端自由项 f(x)的函数句柄；
% a,b 积分上下限，n，求积分的时候，区间[a,b]被等分的个数。
% X,[a,b]上的节点
% Y,与 X 对应的 y(x)的近似值
syms x s
k = ker(x,s); % 得到符号表达式，为了方便公式推导，为转化成第二类 Volterra 积分方程做准备
fx = f(x);
difffx = diff(fx,x);
diffkx = diff(k,x); % 核关于 x 的偏导数
Kxt = - diffkx/ker(x,x); % 得到变换后的第二类 Volterra 积分方程的核
Ft = difffx/ker(x,x); % 得到变换后的第二类 Fredholm 积分方程的自由项
eval(['K = @(x,s)' char(Kxt),';']);
eval(['F = @(x)' char(Ft),';']);
[X,Y] = Volterra2(K,F,lambda,a,b,n);
```

6.8.3 实　例

下面看一个第一类 Volterra 积分方程求解的例子。

【例 6.8-1】 求解第一类 Volterra 积分方程

$$\int_0^x \cos(x-s)y(s)\,ds=\sin(x),\quad 0\leqslant s\leqslant x\leqslant 2$$

其解析解为

$$y(x)=1$$

利用 Volterra1to2 函数求其解，求解过程和结果的代码如下：

```
>> clear
format long
ker = @(x,s) cos(x - s);
f = @sin;
[X,Y] = Volterra1to2(ker,f,1,0,2,10) % 区间[0,2]等分 10 份求解
X =
  Columns 1 through 5
                  0   0.200000000000000   0.400000000000000   0.600000000000000   0.800000000000000
```

```
   Columns 6 through 10
   1.000000000000000  1.200000000000000  1.400000000000000  1.600000000000000  1.800000000000000
   Column 11
   2.000000000000000
Y =
   Columns 1 through 5
   1.000000000000000  0.999933510920748  0.999997182544577  0.999997575205614  0.999989025211731
   Columns 6 through 10
   0.999986433005018  0.999981131172396  0.999978217232382  0.999973581901347  0.999969859080763
   Column 11
   0.999968026966502
```

从上面结果可以看出，数值求解的结果和精确值已经非常接近了。

【例 6.8-2】 求解第一类 Volterra 积分方程：

$$\int_0^x (\sin(x) - s + 2) y(s) \mathrm{d}s =$$

$$16\sin(\sqrt{x}) - 6x\sin(\sqrt{x}) - 16\sqrt{x}\cos(\sqrt{x}) + 2x^2\cos(\sqrt{x}) +$$

$$\sin(x)(2\sin(\sqrt{x}) - 2\sqrt{x}\cos(\sqrt{x}), \quad 0 \leqslant s \leqslant x \leqslant 2$$

其解析解：

$$y(x) = \sin(\sqrt{x})$$

利用 Volterra1to2 函数求其解，求解过程和结果的代码如下：

```
>> clear
format long
ker = @(x,s) sin(x) - s + 2;
f = @(x)16 * sin(x^(1/2)) - 6 * x * sin(x^(1/2)) - 16 * x^(1/2) * cos(x^(1/2)) ...
 + 2 * x^(3/2) * cos(x^(1/2)) + sin(x) * (2 * sin(x^(1/2)) - 2 * x^(1/2) * cos(x^(1/2)));
[X,Y] = Volterra1to2(ker,f,1,0,2,10);
AnalyticValue = sin(sqrt(X));
>> Y,AnalyticValue
Y =
   Columns 1 through 5
                   0  0.439559392317873  0.593543356673783  0.702171643970996  0.781039446118426
   Columns 6 through 10
   0.842624876434972  0.889548477433086  0.926106320336128  0.953559689306749  0.973512470940705
   Column 11
   0.987780814834994
AnalyticValue =
   Columns 1 through 5
                   0  0.432454838953870  0.591127117215293  0.699427870463449  0.779850650384937
   Columns 6 through 10
   0.841470984807897  0.889132044127284  0.925826266699155  0.953580740486920  0.973858565648091
   Column 11
   0.987765945992736
>> plot(X,AnalyticValue,'*')
hold on
```

```
plot(X,Y,'o');hold off
legend('解析解','Volterra1to2 函数求解结果','location','best')
```

从上述结果以及图 6.6 中看以清楚看出，Volterra1to2 函数得到的结果和真实解误差很小。

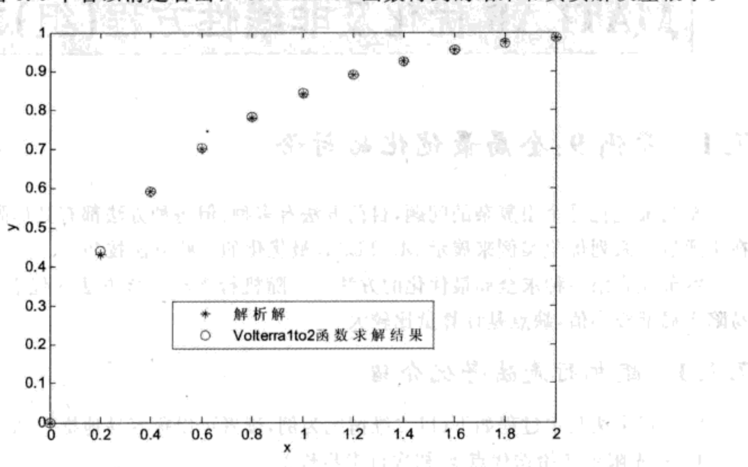


图 6.6　数值解与解析解对比

第 7 章

MATLAB 优化及非线性方程(组)求解案例

7.1 案例 9:全局最优化的讨论

全局最优化是个很复杂的问题,目前方法有多种,但每种方法都有其局限性。本案例目的在于通过一系列优化实例来展示 MATLAB 最优化的一些方法技巧。

本节先介绍一种求全局最优化的方法——随机行走法。该方法的优点是操作简单,不容易陷入局部极小值,缺点是计算量比较大。

7.1.1 随机行走法寻优介绍

随机行走法基本过程如下(以二维情况为例,读者可以很容易地推广到 n 维情形):

1) 首先限定初始迭代点 $\boldsymbol{x}$,初次行走步长 λ。

2) 为了产生合适的迭代控制次数 N,k 置初值 1。

3) 当 $k<N$ 时,随机生成一个$(-1,1)$之间的二维向量 $\boldsymbol{u}$,并标准化 $\boldsymbol{u}$:u/norm(u)。令

```
x1 = x + lamda * (u/norm(u));
```

完成第一步行走。

4) 计算函数值,如果 $f(\boldsymbol{x}_1)<f(\boldsymbol{x})$,即找到一个比初始值好的点,那么 k 重新置 1,回到 2)处。

否则 $k=k+1$,回到 3)处。

5) 如果连续 N 次都找不到更优的值,则认为,最优解就在以当前最优解为中心,当前步长为半径的球内,这时候,我们把 λ 减半,回到 1)处,再开始新一轮行走。

这个过程直到 $\lambda<\varepsilon$ 时结束(ε 是提前设置的一个比较小的数)。

实现基本的随机行走法的 MATLAB 代码如下:

```
function [mx,minf] = randwalk(f,x,lamda,epsilon,N)
% 随机行走法求函数的极小值。输入 f 为所求函数的句柄,
% x 为初始值。lamda 为步长。epsilon 为控制 lamda 的减小的阈值,即 lamda 减小到 epsilon 时
% 迭代停止。
% N 为为了产生较好点的迭代控制次数。
% 函数返回值 mx 为 n 次试验得到的最优解,minf 为响应的最优值。
f1 = f(x(1),x(2));
while(lamda >= epsilon)
    k = 1;
    while(k <= N)
```

```
            u = unifrnd(-1,1,1,2);
            x1 = x + lamda * (u/norm(u));
            f11 = f(x1(1),x1(2));
            if f11 < f1
                f1 = f11;
                x = x1;
                k = 1;
            else
                k = k + 1;
            end
        end
        lamda = lamda/2;
    end
    mx = x1;
    minf = f1;
```

可以利用随机行走法对下述针状函数寻优：

【例 7.1-1】 求下列函数的最大值：

$$f(r)=\frac{\sin(r)}{r}+1 \tag{7-1}$$

其中：

$$r=\sqrt{(x-50)^2+(y-50)^2}+\mathrm{e},\quad 0\leqslant x,y\leqslant 100$$

该函数的三维图形如图 7.1 所示。

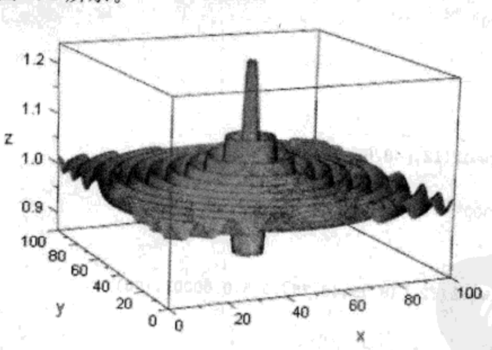


图 7.1 针状函数三维图

从图 7.1 可以看出，该函数是一个多峰函数，在(50,50)处取得全局最大值 1.151 2，第二极大值在其全局最大值附近，一般传统优化方法很容易陷入第二极大值。

经过实际测试表明，当初值[x,y]位于以(50,50)为圆心，半径为大于 1 大于 2 的圆域内，随机行走法几乎都能找到最优点。下面给出该方法和 MATLAB 自带的 fminsearch 的对比。

比较前的准备工作是建立 randwalk 和 fminsearch 各自需要的目标函数的函数句柄：

```
f = @(x) - sin(sqrt( (x(1) - 50)^2 + (x(2) - 50)^2 ) + exp(1))/...
    (sqrt( (x(1) - 50)^2 + (x(2) - 50)^2 ) + exp(1)) - 1;
f2 = @(x,y) - sin(sqrt( (x - 50).^2 + (y - 50).^2 ) + exp(1))./...
    (sqrt( (x - 50).^2 + (y - 50).^2 ) + exp(1)) - 1;
```

比较结果代码如下：

```
% fminsearch 法
>>  [x,fval] = fminsearch(f, [49,49], optimset('TolX',1e-8))
x =
    51.4416 45.2051
fval =
    -1.1284
>> [x,fval] = fminsearch(f, [49.6,49.6], optimset('TolX',1e-8))
x =
    51.6997 45.2904
fval =
    -1.1284
>> [x,fval] = fminsearch(f, [49.65,49.65], optimset('TolX',1e-8))
x =
    50.8348 45.0631
fval =
    -1.1284
>> [x,fval] = fminsearch(f, [49.7,49.7], optimset('TolX',1e-8))
x =
    50.0000 50.0000
fval =
    -1.1511
% 随机行走法
>> [mx,minf] = randwalk(f2,[48.8,48.8],0.5,0.00001,100)
mx =
    50.0000   50.0000
minf =
    -1.1511
>> [mx,minf] = randwalk(f2,[48.75,48.75],0.5,0.00001,100)
mx =
  47.3040   45.7809
minf =
  -1.1284
>> [mx,minf] = randwalk(f2,[48.8,48.75],0.5,0.00001,100)
mx =
  50.0000   50.0000
minf =
  -1.1511
```

可见，randwalk 比 fminsearch 的有效搜索范围要大，fminsearch 在初始值为(49.65，49.65)时按如上默认设置尚且不能迭代到全局最优解(读者如果有兴趣可以自行调整 fmin-

search 函数优化的设置选项，看能否迭代到全局最优解)，而随机行走法在初始值为(48.8，48.75)时就能迭代到最优。

上述是基本的随机行走法，实际上，随机行走法还可以改进，改进后的随机行走法大大提高了全局寻优的能力。

7.1.2　改进的随机行走法寻优

参考 7.1.1 节对基本随机行走法的介绍，在第三步，改进的随机行走法是如下处理的：

随机生成 n 个(－1,1)之间的二维向量 $\boldsymbol{u}$，而不是一个，这样对应 n 个 $\boldsymbol{x}_1$。从这 n 个 $\boldsymbol{x}_1$ 中选取使得 $f(\boldsymbol{x}_1)$最小的 $\boldsymbol{x}_1$ 作为行走目标。这样大大提高了全局寻优的能力，而且对初始值的依赖也大大降低，下面是改进后的代码：

```
function [mx,minf] = randwalknew(f,x,lamda,epsilon,N,n)
% 随机行走法求函数的极小值。输入 f 为所求函数的句柄，
% x 为初始值。lamda 为步长。epsilon 为控制 lamda 的减小的阈值，即 lamda 减小到 epsilon 时
% 迭代停止。
% N 为为了产生较好点的迭代控制次数。
% n 为单步循环行走次数，目的是尽可能走到全局最优点附近
% 函数返回值 mx 为 n 次试验得到的最优解，minf 为响应的最优值。
F = zeros(n,1);
X = zeros(n,2);
f1 = f(x(1),x(2));
while(lamda >= epsilon)
    k = 1;
    while(k <= N)
        u = unifrnd(-1,1,n,2);
        for ii = 1:n
        X(ii,:) = x + lamda * (u(ii,:)/norm(u(ii,:)));
        F(ii) = f(X(ii,1),X(ii,2));
        end
        [f11,kk] = min(F);
        if f11 < f1
            f1 = f11;
            x = X(kk,:);
            k = 1;
        else
            k = k + 1;
        end
    end
    lamda = lamda/2;
end
mx = X(kk,:);
minf = f1;
```

对例 7.1-1 求解的结果如下：

```
>> f2 = @(x,y) - sin(sqrt( (x - 50).^2 + (y - 50).^2 ) + exp(1))./(sqrt( (x - 50).^2 + (y - 50).^2 )
+ exp(1)) - 1;
>> [mx,minf] = randwalknew(f2,[0,0],10,0.00001,1000,10)
mx =
  50.0000 50.0000
minf =
  - 1.1511
>> [mx,minf] = randwalknew(f2,[ - 10,6],10,0.00001,1000,10)
mx =
  50.0000   50.0000
minf =
  - 1.1511
>> [mx,minf] = randwalknew(f2,[2.9765,0.888],10,0.00001,1000,10)
mx =
  50.0000   50.0000
minf =
  - 1.1511
>> [mx,minf] = randwalknew(f2,[12.231,3.777],10,0.00001,1000,10)
mx =
  50.0000   50.0000
minf =
  - 1.1511
>> [mx,minf] = randwalknew(f2,[pi,exp(0.653)],10,0.00001,1000,10)
mx =
  50.0000   50.0000
minf =
  - 1.1511
```

从上面的运行结果可以看出，改进后的随机行走法适应性大大好于基本的随机行走法，求解的初值要求也比基本的随机行走法宽松很多，初值远离[50,50]这个最优解较多的时候，譬如初值为(−10,6)，(0,0)时也能找到最优解。而基本的随机行走法则局限在最优解附近较小的范围内。

下面是一个隐函数寻优的例子，该隐函数即为例 4.2-3 中讨论的隐函数。

【例 7.1-2】 求下列 z 关于 x,y 的隐函数的最小值。

$$z=\sin\left((zx-0.5)^2+2xy^2-\frac{z}{10}\right)\exp\left(-\left((x-0.5-\exp(-y+z))^2+y^2-\frac{z}{5}+3\right)\right) \tag{7-2}$$

其中，$x\in[-1,7]$，$y\in[-2,2]$。

首先建立隐函数的句柄。为了方便读者使用以及验证，代码中所有初值都是随机初值：

```
z = @(x,y)fzero(@(z)z - sin((z * x - 0.5)^2 + x * 2 * y^2 - z/10) * exp( - ((x - 0.5 - exp( - y + z))^2 + ...
    y^2 - z/5 + 3)),rand);
```

读者从下面的测试中可以看出改进的随机行走法的初值选择范围非常具有弹性：

```
z = @(x,y)fzero(@(z)z - sin((z * x - 0.5)^2 + x * 2 * y^2 - z/10) * exp( - ((x - 0.5 - exp( - y + z))^2 + ...
    y^2 - z/5 + 3)),rand);
>> format long
%(由于x,y被限定在[-1,7],[-2,2],因此初值步长设为5左右和区间半径相仿,
%初值随机给出,每步有5个位置供挑选)
tic,[mx,minf] = randwalknew(z,[unifrnd( - 1,7),unifrnd( - 2,2)],5,0.000001,100,5),toc
mx =
   2.898268658270597 - 0.857312901262032
minf =
   - 0.023354083261439
Elapsed time is 9.599392 seconds.
>> tic,[mx,minf] = randwalknew(z,[unifrnd( - 1,7),unifrnd( - 2,2)],...
2.5,0.000001,100,5),toc
mx =
   2.898271925482332 - 0.857312612817192
minf =
   - 0.023354083261446
Elapsed time is 9.556665 seconds.(节省时间,初始步长设为2.5)
>> tic,[mx,minf] = randwalknew(z,[unifrnd( - 1,7),unifrnd( - 2,2)],...
2.5,0.000001,100,3),toc
mx =
   2.898271840049679 - 0.857313122691949
minf =
   - 0.023354083261433
Elapsed time is 5.633099 seconds.(节省时间,每步有3个位置供挑选)
```

从以上测试可以看出,改进后的随机行走法寻优能力大大增强。尤其是每步的随机挑选位置越多,步长越适中,全局寻优的能力就越强,当然,计算时间也越长,但这些都在使用者可控范围内。

【例7.1-3】 Branins's rcos 函数寻优。Branins's rcos 函数表达式如下:

$$f(x,y)=\left(y-\frac{51x^2}{40\pi^2}+\frac{5x}{\pi}-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x)+10,\quad -5\leqslant x\leqslant 10,0\leqslant y\leqslant 15$$

该函数图像可以由下列代码得到:

```
f = @(x,y) (y - (5.1/(4 * pi^2)) * x.^2 + 5/pi * x - 6).^2 + 10 * (1 - 1/(8 * pi)) * cos(x) + 10;
[X,Y] = meshgrid( - 5:0.5:10,0:0.5:15);
fXY = f(X,Y);
surf(X,Y,fXY)
xlabel('x');
ylabel('y');
str = strcat('$ {\left( - \frac{51\, x^2}{40\, {\pi}^2} + \frac{5\, x}{\pi} ',...
    '+ y - 6\right)}^2 - \cos\! \left(x\right)\, \left(\frac{5}{4\, \pi}',...
    '- 10\right) + 10 $');
title(str,'interpreter','latex','fontsize',15)
```

Branins's rcos 函数有三个全局最小值点,取值为0.397 887,分别在(-pi,12.275)、(pi,2.275),(9.424 78,2.475)三处,各种优化方法在求 Branins's rcos 函数全局最优时很容易陷

入局部最优值 0.844 7，有兴趣的读者可以使用 MATLAB 自带的各种优化方法，如遗传算法（相应函数 ga）、模拟退火（相应函数 simulannealbnd）、模式搜索（相应函数 patternsearch）等和优化函数 fminsearch，fmincon，fminunc 尝试一下，会发现比较难找全全局最优点。这里利用随机行走法进行求解，代码如下：

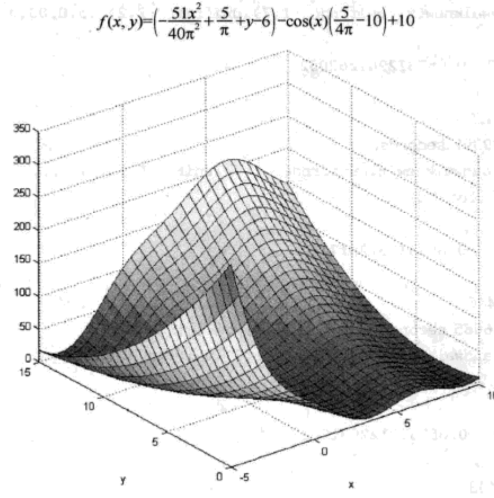


图 7.2 Branins's rcos 函数图像

```
f = @(x,y) (y - (5.1/(4 * pi^2)) * x.^2 + 5/pi * x - 6).^2 + 10 * (1 - 1/(8 * pi)) * cos(x) + 10;
[mx,minf] = randwalk(f,[unifrnd( - 5,10),unifrnd(0,15)],0.5,0.00001,100);
mx =
   - 3.1416   12.2750
minf =
   0.3979
>> [mx,minf] = randwalk(f,[unifrnd( - 5,10),unifrnd(0,15)],0.5,0.00001,100)
mx =
   9.4248   2.4750
minf =
   0.3979
[mx,minf] = randwalk(f,[unifrnd( - 5,10),unifrnd(0,15)],0.5,0.00001,100)
mx =
   15.7080   12.8750
minf =
   0.3979
[mx,minf] = randwalk(f,[unifrnd( - 5,10),unifrnd(0,15)],0.5,0.00001,100)
mx =
   3.1416   2.2750
minf =
   0.3979
```

从上面求解过程来看，基本的随机行走法很快就找全了所有的全局最优解，限于篇幅这里就不再尝试改进的随机行走法了，有兴趣的读者可以自行尝试。可以预见的是，改进的随机行走法也可以轻而易举找到所有的全局最优解。

7.2 案例10:fsolve求非线性方程组的应用

7.2.1 概 述

MATLAB中fsolve函数可以用来求解非线性方程组。fsolve函数功能强大，可以求解很多种类型的非线性方程组，尤其是一些用符号求解函数solve无法求解的方程组。本案例目的在于通过求解一些非线性方程组，帮助读者熟悉fsolve的用法。fsolve函数的调用语法以及简单示例可以参看其帮助文档。这里不再进行叙述。

7.2.2 四元非线性方程组的求解

本小节将用fsolve函数求解一个四元非线性方程组。

【例7.2-1】 求以下非线性方程组的根：

$$\begin{cases} p_0+p_1(1-\exp(-(p_2\cdot(0)^{p_3})))=51.61 \\ p_0+p_1(1-\exp(-(p_2\cdot(9.78)^{p_3})))=51.91 \\ p_0+p_1(1-\exp(-(p_2\cdot(30.68)^{p_3})))=53.27 \\ p_0+p_1(1-\exp(-(p_2\cdot(59.7)^{p_3})))=59.68 \end{cases} \tag{7-3}$$

用fsolve函数可以求解上面的方程，求解代码如下：

```
function p = fsolveDemo1
p0 = [51.61; - 1;0;1];
options = optimset('MaxFunEvals',20000,'MaxIter',2000);
[p,fval] = fsolve(@f,p0,options) ;
    function F = f(p)  % 待求解的方程
        F = [p(1) + p(2) * (1 - exp( - (p(3) * (0)^p(4)))) - 51.61;
            p(1) + p(2) * (1 - exp( - (p(3) * (9.78)^p(4)))) - 51.91;
            p(1) + p(2) * (1 - exp( - (p(3) * (30.68)^p(4)))) - 53.27;
            p(1) + p(2) * (1 - exp( - (p(3) * (59.7)^p(4)))) - 59.68;];
    end
end
```

运行该程序代码，结果如下：

```
  p = fsolveDemo1
Optimization terminated: first - order optimality is less than options.TolFun.
p =
  51.609999999999999
   - 0.465222188130621
   - 0.053730715604489
   0.976132921502588
```

如果我们把 fsolve 的初始值 p0 设为随机的，即把 fsolveDemo1 第二行改为如下代码：

```
p0 = [51.61;rand;unifrnd(-1,1);rand];
```

则运行 fsolveDemo1 多次还可以得到另一组解：

```
p =
  51.609999999999999
  -0.134520891677175
  -0.241198737479646
  0.693446385899554
```

7.2.3 九元非线性方程组的求解

【例 7.2-2】 求以下非线性方程组的根：

$$\begin{cases} 23.3037x_2+(1-x_1x_2)x_3(\exp(x_5(0.485-0.0052095x_7-0.0285132x_8))-1)-28.5132=0 \\ -28.5132x_1+(1-x_1x_2)x_4(\exp(x_6(0.116-0.0052095x_7+0.0233037x_9))-1)+23.3037=0 \\ 101.779x_2+(1-x_1x_2)x_3(\exp(x_5(0.752-0.0100677x_7-0.1118467x_8))-1)-111.8467=0 \\ -111.8467x_1+(1-x_1x_2)x_4(\exp(x_6(-0.502-0.0100677x_7+0.101779x_9))-1)+101.779=0 \\ 111.461x_2+(1-x_1x_2)x_3(\exp(x_5(0.869-0.0229274x_7-0.1343884x_8))-1)-134.3884=0 \\ -134.3884x_1+(1-x_1x_2)x_4(\exp(x_6(0.166-0.0229274x_7+0.111461x_9))-1)+111.461=0 \\ 191.267x_2+(1-x_1x_2)x_3(\exp(x_5(0.982-0.0202153x_7-0.2114823x_8))-1)-211.4823=0 \\ -211.4823x_1+(1-x_1x_2)x_4(\exp(x_6(-0.473-0.0202153x_7+0.191267x_9))-1)+191.267=0 \\ x_1x_3-x_2x_4=0 \end{cases} \tag{7-4}$$

上述方程组比较复杂，难以求其解析解，只能通过数值方法求解。相应的 MATLAB 程序代码如下：

```
function [x,x0,fval,exitflag] = fsolveDemo2
x0 = unifrnd(0,1,1,9); % 生成随机初始值
options = optimset('MaxFunEvals',20000,'MaxIter',5000);
[x,fval,exitflag] = fsolve(@f,x0,options) ;
    function F = f(x)
        F = [23.3037 * x(2) + (1 - x(1) * x(2)) * x(3) * (exp(x(5) * (0.485 - 0.0052095 * x(7) - ...
            0.0285132 * x(8))) - 1) - 28.5132;
            -28.5132 * x(1) + (1 - x(1) * x(2)) * x(4) * (exp(x(6) * (0.116 - 0.0052095 * x(7) + ...
            0.0233037 * x(9))) - 1) + 23.3037;
            101.779 * x(2) + (1 - x(1) * x(2)) * x(3) * (exp(x(5) * (0.752 - 0.0100677 * x(7) - ...
            0.1118467 * x(8))) - 1) - 111.8467;
            -111.8467 * x(1) + (1 - x(1) * x(2)) * x(4) * (exp(x(6) * ( -0.502 - 0.0100677 * x(7) + ...
            0.101779 * x(9))) - 1) + 101.779;
            111.461 * x(2) + (1 - x(1) * x(2)) * x(3) * (exp(x(5) * ( 0.869 - 0.0229274 * x(7) - ...
            0.1343884 * x(8))) - 1) - 134.3884;
            -134.3884 * x(1) + (1 - x(1) * x(2)) * x(4) * (exp(x(6) * (0.166 - 0.0229274 * x(7) + ...
            0.111461 * x(9))) - 1) + 111.461;
```

```
                191.267 * x(2) + (1 - x(1) * x(2)) * x(3) * (exp(x(5) * ( 0.982 - 0.0202153 * x(7) - ...
                0.2114823 * x(8))) - 1) - 211.4823;
                - 211.4823 * x(1) + (1 - x(1) * x(2)) * x(4) * (exp(x(6) * ( - 0.473 - 0.0202153 * x(7) + ...
                0.191267 * x(9))) - 1) + 191.267;
                x(1) * x(3) - x(2) * x(4)];
        end
    end
```

经验证，试验几次后可以求出原方程组的一组解 x，以及相应的初值 x0：

```
x =
  Columns 1 through 4
  0.898509997792005   0.973967802006412   11.648633061077305   10.746159415564769
  Columns 5 through 8
  3.250825125869217   6.710547896318697   - 8.764002776221822   1.251334798194942
  Column 9
  - 0.525131249862975
x0 =
  Columns 1 through 4
  0.847084730088295   0.080752928833763   0.878454584465559   0.706069147286000
  Columns 5 through 8
  0.244023559973308   0.302694211806679   0.195234631882628   0.377503658981705
  Column 9
  0.582932595277113
```

如果我们要求其一组全部大于 0 的解该如何办呢？容易想到的一种办法是用 while 循环，当求得的解不是全部大于 0，或者 fsolve 的第三个返回参数 exitflag 的值不等于 1 时循环一直进行，直到循环条件不满足。这样可以用下列代码求一组全部大于 0 的解：

```
X0 = [];
X = [];
exitflag = 0;
while sum(x > = 0) ~ = 9 | exitflag ~ = 1
    [x,x0,fval,exitflag] = fsolveDemo2;
    if exitflag = = 1
        X0 = [X0;x0];
        X = [X;x];
    end
end
```

用以上代码寻找解，以及全部大于 0 的解。找到一组解就记录下来，同时直到找到全部分量大于 0 的解时退出循环。运行上述代码，循环 10 次找到了三组(其实是两组，有两组解重复)解，下面给出了找到的解以及相应的初值：

```
解 X:
X =
    0.8985   0.9740   11.6486   10.7462   3.2508   6.7105   - 8.7640   1.2513   - 0.5251
    0.8985   0.9740   11.6486   10.7462   3.2508   6.7105   - 8.7640   1.2513   - 0.5251
    0.9000   0.4500   1.0000    2.0001    8.0000   7.9997   5.0000     1.0000   2.0001
```

```
相应的初值 X0：
X0 =
    0.3655  0.4817  0.9437  0.1787  0.2950  0.4509  0.6806  0.2808  0.4612
    0.6870  0.3019  0.7283  0.5213  0.7212  0.8447  0.4929  0.2533  0.9737
    0.6211  0.9356  0.1610  0.2202  0.6481  0.9630  0.2586  0.5750  0.0260
```

7.2.4 非线性积分方程的求解

在第 6 章我们讨论了 Fredholm 积分方程以及 Volterra 积分方程这样一些线性积分方程的求解方法，即未知函数 $y(x)$ 不论在积分号下还是在积分号外都是一次的，而实际中还有大量的非线性积分方程。非线性积分方程是指未知函数 $y(x)$ 不是一次的，一些非线性积分方程可以通过求解线性积分方程的数值解法来求解。只不过这时求解的不是线性方程，而是求解非线性方程了。这时可以利用求解线性积分方程的方法结合 fsolve 函数进行求解。这里作为示例，求解一个第二类非线性 Volterra 积分方程。

【例 7.2-3】 求解下列非线性 Volterra 积分方程：

$$y(x)=\int_0^x (xs-1)y^2(s)\mathrm{d}s-\frac{1}{4}x^5-\frac{2}{3}x^4-\frac{1}{6}x^3+x^2+2x+1,\quad 0\leqslant s\leqslant x\leqslant 1$$

其解析解为

$$y(x)=x+1$$

采用 Newton－Cotes 积分方法离散上述方程，并利用 fsolve 求解积分节点上的 y 值，相应的 MATLAB 程序代码如下：

```
function [X,Y,fval] = NonlinearVolterra2(ker,f,lambda,FY,a,b,n,Y0)
 % 利用 Newton - Cotes 积分公式求解第二类非线性 Volterra 积分方程：
 % y(x) = lambda * int(k(x,s) * FY(y(s)) * ds,a,x) + f(x),a<= s<= x<= b
 % ker,核函数 k(x,s)的函数句柄；f,自由项 f(x)的函数句柄；lambda,参数；
 % FY,y(s)的非线性泛函句柄，譬如(y(s))^2,则 FY = @(y) y^2;a 积分下限,[a,b],x 的取值范围
 % n,求积分的时候，区间[a,b]被等分的个数。Y0,求解 Y 的时候初值向量，长度等于 n
 % X,[a,b]上的节点
 % Y,与 X 对应的 y(x)的近似值
 % fval,求解非线性方程组的时候的函数值，理想情况下应该严格等于 0
if length(Y0) ~= n + 1
    error(['Y0 的长度必须要等于 n + 1:',num2str(n + 1)]);
end
Y0 = Y0(:);
eval(['ker = ',vectorize(ker),';']); % ker 统一为点乘形式
eval(['f = ',vectorize(f),';']); % f 统一为点乘形式
eval(['FY = ',vectorize(FY),';']); % FY 统一为点乘形式
if nargin <= 5
    % 没有指定 n 的话，n 为 5
    n = 5;
end
A = cell(n,1); % 存储 Cotes 系数
for k = 1:n
    A{k} = NewtonCotes(k);
```

```
end
K = zeros(n+1);
h = (b-a)/n;
xk = (a:h:b)';
for ix = 2:n+1
    for jx = 1:ix
        K(ix,jx) = (ix-1) * h * A{ix-1}(jx) * ker(xk(ix),xk(jx)); % 计算 K 矩阵
    end
end
K = lambda * K;
F = f(xk); % 得到自由项 f(x)关于求积节点的函数值向量
if isscalar(F) % 针对某些自由项函数为常数的情形
    F = F * ones(size(xk));
end
X = xk'; % 对应的自变量 X 的值
options = optimset('TolFun',1e-16); % 非线性方程求解设置,函数值改变容差 1e-16
% 利用 fsolve 函数求解离散原方程后得到非线性方程组
[Y,fval] = fsolve(@(Y) K * FY(Y) + F - Y,Y0,options);
Y = Y';
fval = fval';
end
%==========================================
% 求 Cotes 系数的函数(普通子函数形式)
%==========================================
function A = NewtonCotes(n)
% n 等分求积区间时,返回 Cotes 系数
syms t;
A = zeros(n+1,1);
for k = 0:n
    % 利用符号计算得到 Cotes 系数中被积函数的表达式
    symfun = prod(t-[0:k-1,k+1:n]);
    % 将得到的被积函数表达式转成"点乘"形式并生成相应的函数句柄
    eval(['fun = @(t)',vectorize(symfun),';']);
    A(k+1) = quadgk(fun,0,n) * (-1)^(n-k)/gamma(k+1)/gamma(n-k+1)/n;
end
end
```

利用上述程序求解例 7.2-3,代码如下:

```
>> ker = @(x,s) x * s - 1; % 核函数
f = @(x) polyval([-1/4,-2/3,-1/6,1,2,1],x); % 自由项函数
lambda = 1;
FY = @(Y) Y.^2; % y(x)的非线性泛函
a = 0;
b = 1;
n = 10; % 10 等分[a,b]区间
Y0 = ones(1,n+1); % 求解 Y 的初始值
[X,Y,fval] = NonlinearVolterra2(ker,f,lambda,FY,a,b,n,Y0)
Optimization terminated: relative function value changing by less
```

```
than max(options.TolFun^2,eps) and sum - of - squares of function
values is less than sqrt(options.TolFun).
X =
    Columns 1 through 8
         0 0.1000  0.2000  0.3000  0.4000  0.5000  0.6000  0.7000
  Columns 9 through 11
  0.8000  0.9000  1.0000
Y =
  Columns 1 through 8
  1.0000  1.0999  1.2000  1.3000  1.4000  1.5000  1.6000  1.7000
  Columns 9 through 11
  1.8000  1.9000  2.0000
fval =
  1.0e - 015 *
  Columns 1 through 8
  0   - 0.2220  0 0.2220  0 0 0.2220  0
  Columns 9 through 11
  - 0.2220  0.2220  0.4441
```

从求解出的结果来看，其和解析解相差不大。读者可以参考上述程序求解其他类型的非线性积分方程。需要注意的是初始值 Y0 的选取，如果返回的 fval 和 0 相差较大，则要改变初始值 Y0，多尝试几次。

7.3 案例 11：渐变光波导方程求解

7.3.1 求解渐变光波导的模方程

【例 7.3-1】 求以下非线性方程的根：

$$\int_{1\times10^{-10}}^{xc} k\sqrt{\left[ns+dN\times\frac{1}{\pi}\left(\frac{x}{D}\right)\exp\left[-\left(\frac{x}{D}\right)^2\right]\int_{-0.5}^{0.5}\frac{\exp(-\xi^2)}{\left(\frac{x}{D}\right)^2+\xi^2}\mathrm{d}\xi\right]^2-N^2}\,\mathrm{d}x = \frac{\pi}{4}+\tan^{-1}\left[\sqrt{\frac{N^2-(nc)^2}{N_1^2-N^2}}+\frac{k^2N_1\times-(1.8558)\times10^4}{2\left(\sqrt{k^2N_1^2-k^2N^2}\right)^3}\right] \tag{7-5}$$

其中：

ns=1.520167；

nf=1.548167；

$k=2\pi/0.6328\mathrm{e}-6$；

nc=1；

D=2e−6；

N_1=1.5436744；

dN=0.0235072。

该方程是求解渐变光波导的模方程，形式比较复杂，N 是未知数，要想求解 N 得计算等式

左侧的积分，这么复杂的积分只能通过数值方法求解。结合本书第 6 章介绍的 quadl＋arrayfun 的方法，可以写出求解代码如下：

```
function N = example731(xc)
ns = 1.520167;
nf = 1.548167;
k = 2 * pi/0.6328e - 6;
nc = 1;
D = 2e - 6;
N1 = 1.5436744;
dN = 0.0235072;
% 用匿名函数形式表示关于 N 的复杂非线性方程
ff = @(N) quadl(@(y) k * sqrt((ns + dN * y/(pi * D). * exp( - (y/D).^2). * arrayfun(@(y) ...
    quadl(@(s) exp( - s.^2)./(s.^2 + (y/D).^2), - 0.5,0.5),y)).^2 - N.^2),1e - 10,xc)...
    - pi/4 - atan(sqrt((N.^2 - nc^2)/(N1^2 - N.^2)) + (k^2 * N1 * ( - 1.855800789988048e4))/...
    ((k^2 * N1^2 - k^2 * N.^2)^1.5 * 2));
N = fzero(ff,[1.52017,1.5246095]);
```

对于 xc＝1.537433e－6，运行上述代码，结果如下：

```
>> format long
>> N = example731(1.537433e - 6)
N =
  1.524606031629952
```

需要说明的是，fzero 求 ff 的解时，区间[1.52017，1.5246095]是按下述方法确定的：

因为 ff 在 N＝1.52 时大于 0，而 ff 在 N＝1.543 时值是虚数，所以 fzero 的求解区间右侧不能包含 1.543，具体方法就是用 ff 函数试验 1.543 附近的值，直到区间右侧的值是负数，或者实部是负数，虚部很小为止。区间[1.52017，1.5246095]就是这样得到的。

关于含有积分的非线性方程求解，7.3.2 节介绍的例子也是不错的一个实例。

7.3.2　二维渐变光波导方程作图

【例 7.3－2】 对下述方程作 Δn 与 D 的关系图。

$$\int_0^{x_0} \sqrt{k^2 n^2(x) - \beta^2}\,\mathrm{d}x =$$

$$\frac{\pi}{4} + \tan^{-1}\left[\frac{\sqrt{\beta^2 - k^2}}{\sqrt{k^2(n_s + \Delta n)^2 - \beta^2}} - \frac{\Delta n k^2 (n_s + \Delta n)}{\sqrt{\pi} D \left(\sqrt{k^2(n_s + \Delta n)^2 - \beta^2}\right)^3}\right] \qquad (7-6)$$

其中：

$$n(x) = n_s + \Delta n \cdot \mathrm{erfc}\left(\frac{x}{D}\right) \qquad (7-7)$$

$n(x_0)=1.554\,336$，$k=\frac{2\pi}{\lambda}=\frac{2\pi}{632.8\times10^{-9}}$，$\beta=1.554\,336k$，$\Delta n$ 的取值范围为[0.04，0.08]。

这个方程是针对渐变平面光波导的，也叫二维渐变光波导。其中第一个公式 $n(x)$ 是它的折射率分布，n_s 是基板折射率，Δn 是折射率增量。下面给出 D 关于 Δn 的函数代码：

```
function y = mytdraw(x)
N0 = 1.554336; % 赋值,x - deltan y - D
k = 2 * pi/(632.8e - 9);
B0 = k * N0;
ns = 1.521145;
eq1 = @(y) quadl(@(z) k * sqrt((ns + x. * erfc(z./y)).^2 - N0^2),...
    0,y * erfcinv((N0 - ns)/x)) - pi/4 - atan(sqrt((B0^2 - k^2)/(k^2 * (ns + x)^2 - B0^2)) - ...
    x * (k^2) * (ns + x)/(pi^0.5 * y * (k^2 * (ns + x)^2 - B0^2)^1.5));
y = fzero(eq1,0.000001);
end
```

画图代码如下：

```
clear all
x = linspace(0.04,0.08,801); % x赋值 挨个画图
y = zeros(size(x));
for i = 1:length(x)
y(i) = mytdraw(x(i));
end
plot(x,y)
xlabel('\Deltan');ylabel('y');
```

画出的图形如图7.3所示。

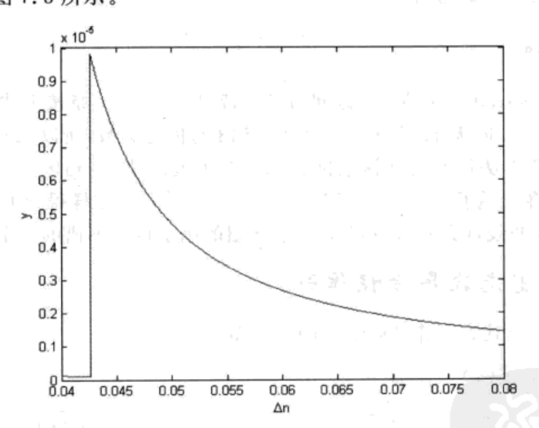


图7.3 Δn与D的关系图

7.4 案例12：遗传算法在复杂系统可靠度和冗余度分配优化中的应用

7.4.1 问题提出

随着科学技术和社会的不断发展进步，人们需要解决生产、生活中的问题的规模以及难度越来越大，这时往往需要一系列功能模块之间的相互配合才能完成。这些模块之间的功能有



的是相同的，有的是缺一不可的。这些模块有效组织在一起构成复杂的系统。

复杂系统的可靠度依赖于各子系统和模块的可靠度，而提高子系统和模块的可靠度必然引起系统总造价增加，这种依赖关系是非线性的。在一定可靠度下，提高系统模块的冗余数将引起系统造价大幅上升，并且由于成本造价以及先天等条件的限制，单独依靠提高某个模块可靠度来提高控制系统可靠度是困难的。因此，系统可靠度的最优分配非常重要。保证子系统具有优化配置的可靠度指标可以通过提高模块的可靠度指标和增设冗余模块实现。然而，提高子系统可靠度指标和增设冗余模块，哪种方式更经济是设计中需要考虑的重要问题。一般情况下，最优冗余个数与子系统的可靠度指标相关。因此，模块的可靠度指标和子系统冗余数应同时优化，以便在满足约束范围内使整个系统的可靠度最优。换句话说，就是要求系统设计者不仅要确定冗余数，而且还必须确定每个模块的可靠度。这是一个模块可靠度指标和冗余数一体化并发设计的问题。

本案例研究复杂系统可靠度和冗余度分配优化问题，建立子系统可靠度指标和冗余个数一体化并发设计问题的提法，将遗传算法应用于复杂系统可靠度和冗余数的最优分配这类非线性连续变量和离散变量混合问题的求解，并给出相应的 MATLAB 代码。

7.4.2　数学模型

很多复杂系统可以看成是混合串-并联系统，即含有 n 个子系统，每个子系统又含有 m 个并联模块的混合串-并联系统。

提高系统可靠度的主要方法是提高元部件的可靠度和在系统中采用复式备份结构，这两种方法在提高系统可靠度的同时，增加了系统的投资、权量和体积。因此系统可靠度的优化设计应该在满足资金、结构、性能等条件的约束下，提高系统的可靠度；或者在满足一定可靠度要求时，尽量减少投资，取得最大经济效益。这两种可靠度优化问题的数学表达为两种。

1）要求系统的可靠度最大，即

$$R_{\max} = \prod_{i=1}^{n} R_i(m_i) \tag{7-8}$$

且满足一些约束条件：

$$\sum_{i=1}^{n} w_{li}(m_i) \leqslant W_l, \quad l = 1,2,\cdots,k \tag{7-9}$$

式中，$w_{li}(m_i)$ 表示第 l 个子系统中第 i 个模块关于约束条件 l 的数量；W_l 为系统关于约束条件 l 的总数量。

系统可靠度 R 一般是 m_i 的非线性函数。

2）要求系统的投资最小，即

$$C_{\min} = \sum_{i=1}^{n} c_i(m_i) \tag{7-10}$$

满足系统可靠度不低于 R_r

$$R = \prod_{i=1}^{n} R_i(m_i) \geqslant R_r \tag{7-11}$$

且满足其他要求的约束条件：

$$\sum_{i=1}^{n} w_{li}(m_i) \leqslant W_l, \quad l = 1,2,\cdots,k \tag{7-12}$$

带有可选设计和冗余度系统的可靠度。

假定系统包含 N 个子系统，对于每个子系统，系统设计者有多个可选设计来满足可靠性要求。本案例中该问题可以描述为如下非线性整数规划：

$$R(\theta)_{\max} = R(m,\alpha) = \prod_{i=1}^{N} R_i(m_i,\alpha_i) \tag{7-13}$$

s. t.

$$C(m,\alpha) = \sum_{i=1}^{N} c_i(\alpha_i)m_i \leqslant C \tag{7-14}$$

$$W(m,\alpha) = \sum_{i=1}^{N} w_i(\alpha_i)m_i \leqslant W \tag{7-15}$$

$$1 \leqslant m_i \leqslant u_i ; 1 \leqslant \alpha_i \leqslant \beta_i$$

式中，θ 为待优化的参数向量；$\alpha_i(i=1,2,\cdots,N)$ 为第 i 个子系统可选用的设计；$m_i(i=1,2,\cdots,N)$ 为第 i 个子系统相同的冗余单元数；u_i 为第 i 个子系统相同的冗余单元上界；β_i 为第 i 个子系统可选用设计的上界；$R(m,\alpha)$ 为具有冗余单元 m 和可选设计 α 的系统可靠度；$C(m,\alpha)$ 为系统满足冗余单元 m 和可选设计 α 的费用函数；$W(m,\alpha)$ 为系统满足冗余单元 m 和可选设计 α 的权重函数；C 为系统费用的总约束；W 为系统权重的总约束。

对串-并联冗余系统的子系统 i，有

$$R_i(m_i,\alpha_i) = 1 - [1 - R_i(\alpha_i)]^{m_i} \tag{7-16}$$

7.4.3 遗传算法简介

遗传算法 (Genetci Algorithm，AG)是模拟自然界生物进化机制的一种算法，即遵循适者生存、优胜劣汰的法则，也就是寻优过程中有用的保留，无用的则去除。在科学和生产实践中表现为，在所有可能的解决方法中找出最符合该问题所要求的条件的解决方法，即找出一个最优解。

遗传算法最初的目的是研究自然系统的自适应行为，并设计具有自适应功能的软件系统。它的特点是对参数进行编码运算，不需要有关体系的任何先验知识，沿多种路线进行平行搜索，不会落入局部较优的陷阱，能在许多局部较优中找到全局最优点，是一种全局最优化方法。

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于很多学科。

基本的遗传算法的流程如图 7.4 所示。

下面对流程图各个步骤的意义以及完成的工作进行说明：

1) 设计染色体编码方式。就是将问题的解用一种码来表示，从而将问题的状态空间与遗传算法的码空间相对应，这在很大程度上依赖于问题的性质，并将影响遗传操作的设计。由于遗传算法的优化过程不是直接作用于问题参数本身，而是在一定编码机制对应的码空间上进行的，因此编码的选择是影响算法性能与效率的重要因素。针对函数优化的编码技术主要有二进制编码、十进制编码、实数编码等。二进制编码将问题的解用一个二进制 0-1 字符串来表示；十进制编码将问题的解用一个十进制串来表示；等等

2) 随机生成初始群体。遗传算法是群体型操作算法，在对解空间变量进行编码后，紧接着就要随机产生 N 个遵循这种编码的个体(染色体)，构造遗传算法的初始群体，然后以这个初始群体作为起始点开始按代搜索。

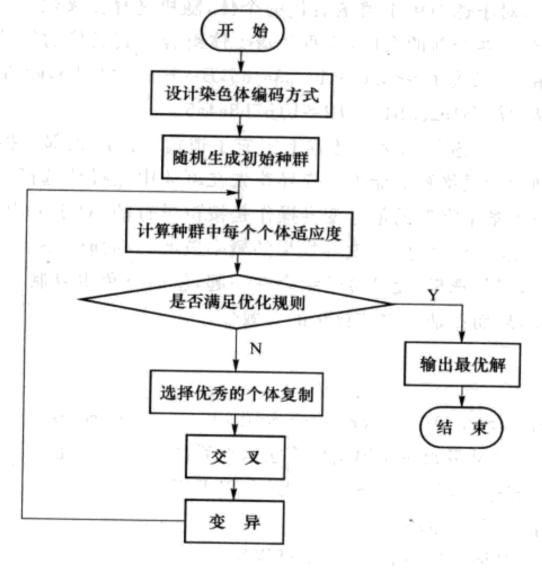


图 7.4　遗传算法流程图

3) 计算种群中每个个体适应度。遗传算法在搜索进化过程中一般不需要其他外部信息，仅用适应度函数值来评估个体或解的优劣，并作为以后遗传操作的依据。适应度函数表明个体对环境适应能力的强弱，不同的问题，适应度函数的定义方式也不同。譬如，对于简单的最小化问题，通常可以直接将目标函数变换成适应度函数，可以将个体 x 的适应度 $f(x)$ 定义为 $M-C(x)$ 或 $e^{-aC(x)}$，其中 M 为一足够大正数，$C(x)$ 为个体的目标值，$a>0$。对于复杂优化问题，往往需要构造合适的评价函数，使其适应问题的特点，方便遗传算法进行优化。

由于适应度度量意义下的个体差异与目标函数值度量意义下的个体差异有所不同，因此若适应度函数设计不当，将难以体现个体的差异，选择操作的作用就很难体现出来，从而造成早熟收敛等问题。对适应度进行调节是常用的改进方法，如线性变换和指数变换等，即通过某种变换改变原适应度间的比例关系。

4) 是否满足优化规则，这个规则是事先制定的。由于遗传算法可以无限制进化下去，初始种群进化到一定程度后往往很难再有大的改善，这时候就需要决定是否停止进化。优化规则的制定有多种，譬如进化到一定的代数、得到的解使得目标函数值达到某个条件等都可以作为优化规则。

5) 选择优秀的个体复制(selection)，又称复制或繁殖(reproduction)。选择操作的目的是为了从当前群体中选出生命力强的染色体，使它有机会保留用以繁殖后代。判断染色体优良与否的准则就是各自的适应度值，个体适应度值越大，其被选择的机会就越多。选择操作体现了达尔文的优胜劣汰、适者生存原则。选择的方法根据不同的问题，采用不同的方案，最常见的方法有比率法、排列法和比率排列法。

6) 交叉(crossover)，又称重组(recombination)或配对(breeding)。交叉操作是遗传算法

中最主要的遗传操作，对于选中用于繁殖后代的个体，随机选择交叉位置 k，交换两个基因串位置 k 右边的部分，产生两个新的个体，这两个新个体组合了其父代的特性。例如，有两个二进制编码的个体 A 和 B，长度 $L=5$，$A=$a1a2a3a4a5，$B=$b1b2b3b4b5，随机选择一个整数 $k=4$，A 与 B 经交叉后为 $Al=$a1a2a3b4b5，$Bl=$b1b2b3a4a5。

7）变异(mutatinon)。选择和交叉基本上完成了遗传算法的大部分搜索功能，而变异则增加了遗传算法找到接近最优解的能力。变异首先在群体中随机地选择一个个体，以一定的概率随机改变基因串中某个字符的值。变异操作是按位进行的，对于采用二进制编码的个体来说，就是将1变为0或将0变为1。变异发生的概率极低(一般取值为0.001～0.5之间)，它本身是一种随机搜索，但与选择、交叉算子结合在一起，就能避免由复制和交叉算子引起的某些信息的永久性丢失，从而保证了遗传算法的有效性。

7.4.4 实例分析

这里以长输油管道 SCADA(supervisory control and data acquisition)系统，即数据采集与监视控制系统为例，来说明如何应用遗传算法来对系统可靠度和冗余度进行分配优化。

在对 SCADA 系统进行可靠度优化设计前，作以下假设：

1）系统中采用的冗余为主动冗余。

2）每一个子系统都是相互独立，且是不可维修的。

3）每一个子系统有两个状态，即工作状态和故障状态。

4）每一个子系统的可靠度为常数。

这样，SCADA 系统的优化设计问题可描述为：寻找最优的各可选设计子系统 $\alpha_1,\alpha_2,\cdots,\alpha_n$，和子系统冗余数 $m_1,m_2,\cdots,m_n$，以使系统在满足系统总造价和总权重条件下可靠度 R 最高。

SCADA 系统是具有冗余单元和可选设计的混合串-并联系统，适合于下列可靠性数学模型：

$$R(\theta)_{\max}=R(m,\alpha)=\prod_{i=1}^{N}\{1-[1-R_i(\alpha_i)]^{m_i}\} \tag{7-17}$$

s. t.

$$C(m,\alpha)=\sum_{i=1}^{N}c_i(\alpha_i)m_i\leqslant C \tag{7-18}$$

$$W(m,\alpha)=\sum_{i=1}^{N}w_i(\alpha_i)m_i\leqslant W \tag{7-19}$$

$$1\leqslant m_i\leqslant u_i;1\leqslant\alpha_i\leqslant\beta_i$$

$$\text{其中}:u_i=2,\beta_i=3$$

式中，子系统可靠度 R_i 是[0,1]之间的值，而冗余数 $m_i(i=1,2,\cdots,N)$ 只能取整数。因此，上面优化问题实际上是连续变量和离散变量并存的混合非线性规划问题。

一般情况下，长输油管道 SCADA 系统包含有站控 PCL 系统、调节阀子系统、微波通信系统、变电所子系统、中心控制系统、泄漏检测系统六个子系统。本例是一个含有六个子系统的 SCADA 系统，每个子系统有三种可选设计，每一子系统的冗余数最多为2，所有子系统的可选设计数据如表7-1所列。

表 7-1 冗余单元与可选设计数据表

子系统 i	可选设计								
	1			2			3		
	R	c_i	w_i	R	c_i	w_i	R	c_i	w_i
1	0.91	40	9	0.92	41	8	0.91	39	10
2	0.90	65	10	0.94	54	10	0.92	57	14
3	0.92	60	12	0.91	55	11	0.90	62	10
4	0.93	64	13	0.90	62	10	0.93	66	12
5	0.90	68	13	0.91	70	12	0.91	67	14
6	0.92	48	12	0.92	51	14	0.93	49	15

约束条件如下：

$$C(m,\alpha)=\sum_{i=1}^{6}c_i(\alpha_i)m_i\leqslant 670 \qquad (7-20)$$

$$w(m,\alpha)=\sum_{i=1}^{6}w_i(\alpha_i)m_i\leqslant 144 \qquad (7-21)$$

所用的遗传算法步骤如下：

1) 根据可靠性数据给出优化参数向量 $\boldsymbol{\theta}=[m_1,\alpha_1,m_2,\alpha_2,\cdots,m_N,\alpha_N]^{\mathrm{T}}$ 的解域范围，参数向量所有分量均为整数型。

2) 确定种群规模、交叉概率、变异概率、最大进化代数：种群规模数为 $S=50$；交叉概率 $P_c=0.9$，变异概率 $P_m=0.01$；最大进化代数取 $G=200$。

3) 种群初始化：采用整数值编码，在解的可行域随机确定 50 个个体(可能的解)构成初始种群。

4) 计算每一个个体的适应度：适应度函数采用如下线性函数：

$$fit(P)=2-sp+2(sp-1)\frac{(P-1)}{(N_i-1)} \qquad (7-22)$$

式中，N_i 为种群的大小；P 为根据目标函数的大小所确定的个体在种群中的位置，譬如，本例目的是最大化目标函数，因此，目标函数值最小的个体，其 P 值为 1，而目标函数值最大的个体，其 $P=N_i$，sp 为选择压力，$1\leqslant sp\leqslant 2$，一般取 $sp=1.75$。对每一个个体，计算其是否违反约束条件，如果是，则个体的适应度累加上一个与违反约束条件程度成正比的正值；否则，个体的适应度保持不变。

5) 选择(繁殖)操作：将种群中的个体适应度由大到小排序，然后根据单个个体所对应的适应度确定其繁殖后在交配池中所占的比例

$$R_i=\frac{fit(i)}{\sum_{i=1}^{N_i}fit(i)} \qquad (7-23)$$

6) 交叉操作：交叉操作是遗传算法中最主要的遗传操作。这里采用离散交叉操作方式产

生后代。具体操作方式如下：

$$\theta_i^0=\theta_i^{P_{a1}}\cdot a_i+\theta_i^{P_{a2}}\cdot(1-a_i)\quad i=1,2,\cdots,n \tag{7-24}$$

式 7-24 中，$\theta_i^{P_1}$，$\theta_i^{P_2}$ 分别为父代 P_{a1}，P_{a2}的第 i 个分量；θ_i^0 为产生的子代的第 i 个分量；$a_i\in\{0,1\}$，为 0-1 离散均匀分布的随机变量。

7) 变异操作：对种群中每一个体以变量概率 P_m 随机改变某一分量的值。改变方式可以灵活设定，例如，某一分量可取 k 个值，变异的时候可以在剩下的 $k-1$ 个值里随机挑一个替换掉原来的。

8) 种群是否达到最大进化代数，如果未达到则转向步骤 4)，否则此时种群中适应度最大的个体所对应的目标函数值，即为全局最优解。

根据上述步骤可以编写求解问题的 MATLAB 代码如下：

```
function [x,f] = GAforSCADA
S = 50;
Pc = 0.90;
Pm = 0.01;
G = 200;
sp = 1.75;
R = [.91 .92 .91;.90 .94 .92;.92 .91 .90;.93 .90 .93;.90 .91 .91;.92 .92 .93];
c = [40 41 39;65 54 57;60 55 62;64 62 66;68 70 67;48 51 49];
w = [9 8 10;10 10 14;12 11 10;13 10 12;13 12 14;12 14 15];
% 向量化的目标函数
ObjectFun = @(m,a) prod(1 - (1 - R(sub2ind([6,3],repmat(1:6,S,1),a))).^m,2);
fit = @(P) (2 - sp + 2 * (sp - 1) * (P - 1)/(S - 1)); % 适应度函数
%% 产生初始种群
theta(:,2:2:12) = unidrnd(3,S,6);
theta(:,1:2:11) = unidrnd(2,S,6);
%% 进化 200 次
f = zeros(G,1); % 每一代的最优可靠度
x = zeros(G,12); % 每一代的最优个体
for g = 1:G
    %% 计算适应度并令违反约束函数的适应度为 0
    % 计算各个个体的目标函数并排序
    [sort0,P] = sort(ObjectFun(theta(:,1:2:11),theta(:,2:2:12)));
    alpha = theta(P(end),2:2:12);
    m = theta(P(end),1:2:11);
    if g > 1&&(sum(c(sub2ind([6,3],1:6,alpha)). * m)> 670 || ...
        sum(w(sub2ind([6,3],1:6,alpha)). * m)> 144 || sort0(end)< f(g - 1))
    f(g) = f(g - 1);
    x(g,:) = x(g - 1,:);
else
    f(g) = sort0(end);
    x(g,:) = theta(P(end),:);
end
FIT = zeros(S,1);
FIT(P) = fit(1:S); % 各个体的适应度;
for k = 1:S
```

```
        alpha = theta(k,2:2:12);
        m = theta(k,1:2:11);
        if ( sum(c(sub2ind([6,3],1:6,alpha)). * m)> 670 || ...
                sum(w(sub2ind([6,3],1:6,alpha)). * m)> 144 )
            FIT(k) = 0; % 令违反约束函数的适应度为 0
        end
    end
    %% 该部分是复制操作
    CopyP = FIT/sum(FIT); % 各个体在交配池中的比例
    Copyind = find(CopyP > 0); % 记录下比例大于 0 的,也就是不被淘汰的个体
    % 01 区间划分(各区间长度依次等于各个体在交配池中所占比例)
    Interval01Divide = cumsum(CopyP(Copyind));
    Test = rand(S,1); % 生成 50 个 0,1 随机数
    % 随机数落入的区间序号,表示该区间代表的个体被选中
    TestResult = arrayfun(@(x) find(Interval01Divide > = x,1,'first'),Test);
    % 各个体选中的次数,进而决定各个体复制的次数
    Frequency = accumarray(TestResult,1);
    Theta = cell(length(Frequency),1);
    for kk = 1:length(Frequency)
        % 根据 Frequency 每个分量决定相应个体的复制次数
        Theta{kk} = repmat(theta(Copyind(kk),:),Frequency(kk),1);
    end
    Theta = cell2mat(Theta);
    %% 该部分是交叉操作
    RandMatch = randperm(S); % 随机配对
    for kk = 1:S/2
        if rand > 1 - Pc % 满足交叉概率,下面交叉
            a = unidrnd(2,1,12) - 1;
            temp = Theta(RandMatch(2 * kk - 1),:). * a + Theta(RandMatch(2 * kk),:). * (1 - a);
            Theta(RandMatch(2 * kk),:) = Theta(RandMatch(2 * kk - 1),:). * ...
                (1 - a) + Theta(RandMatch(2 * kk),:). * a;
            Theta(RandMatch(2 * kk - 1),:) = temp;
        end
    end
    %% 该部分是变异操作
    for kk = 1:S
        if rand < Pm % 满足变异概率
            pos = unidrnd(12,1); % 生成变异位置(12 个位置中的随机一个)
            if mod(pos,2) == 1 % 位置是奇数,表示 m 参数
                % 将 m 的解集合{1,2}去掉当前的 m 值,剩下的那个覆盖原来的 m
                Theta(kk,pos) = setdiff([1,2],Theta(kk,pos));
            else % 偶数位置,表示 a 参数
                % 将 a 的解集合{1,2,3}去掉当前的 a 值
                temp = setdiff([1 2 3],Theta(kk,pos));
                % 剩下的两个值中随机挑一个覆盖原来的 a 值
                Theta(kk,pos) = temp(unidrnd(2));
            end
        end
    end
    theta = Theta;
```

```
end
plot(1:G,f,'linewidth',3);
xlabel('迭代次数');
ylabel('系统可靠度');
title('迭代次数同系统可靠度曲线')
```

运行

```
[x,f] = GAforSCADA;
```

可以得到进化过程中 200 代，每一代的最优个体 x 以及相应的系统适应度 f。经过多次运行该函数，发现函数寻优比较稳定，基本上每次都能得到如下最优解：

$$\theta^* = [(2,2),(2,2),(2,1),(2,1),(2,3),(2,3)]$$

上述解对应表 7－2 所列内容。

表 7－2　θ^* 对应的可选设计和冗余度分配表

子系统编号	可选设计号	冗余数	子系统可靠度
1	2	2	$1-(1-0.92)^2=0.9936$
2	2	2	$1-(1-0.94)^2=0.9964$
3	1	2	$1-(1-0.92)^2=0.9936$
4	1	2	$1-(1-0.93)^2=0.9951$
5	3	2	$1-(1-0.91)^2=0.9919$
6	3	2	$1-(1-0.93)^2=0.9951$

对应的系统可靠度 $R(\theta^*)=0.9936^2\times0.9951^2\times0.9964\times0.9919=0.9662$。

整个遗传算法进化过程中的进化代数（迭代次数）同系统可靠度之间的变化曲线如图 7.5 所示（由于遗传算法的随机性，每次运行曲线不会完全一样，但是绝大多数都会最终达到最优解）。

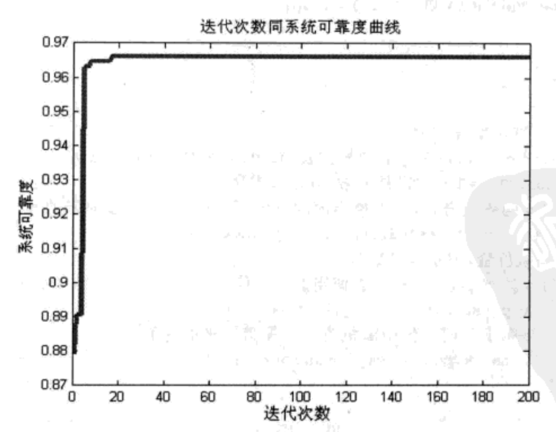


图 7.5　迭代次数同系统可靠度曲线

从图 7.5 中的曲线可以看到,遗传算法在进化了不到 50 代后便趋于稳定了,此时已经求出了最优解。

作为遗传算法寻优的应用,在 7.5 节将介绍一个稍微复杂的案例。

7.5 案例 13:遗传算法在车间设备布局优化中的应用

7.5.1 问题提出

在全球制造业竞争日益激烈的今天,一个高效的低成本的制造系统对于增强企业竞争力来说是至关重要的,其中车间设备布局是制造系统设计的一个重要内容。设备是企业进行生产的基本单元,合理的设备布局对均衡设备能力、保持物流平衡、降低生产成本起着至关重要的作用。有关研究表明,大约 20%以上的加工费用用于物料运输,而合理的设备布局至少能降低 10%~30%的物料运输费用。

车间设备布局问题是一种组合优化问题,具有非线性,NP(Net Profit 的缩写,即净利润,也称税后利润或净收入)难等特性。车间设备平面布局问题可抽象为一维布局和二维布局问题。其中,一维布局可被看做单行设备布局问题;二维布局可归结为多行设备布局问题。求解车间布局优化问题,传统的优化算法一般很难奏效。目前,已经有许多学者应用遗传算法来解决车间设备布局问题并取得了一定的成果,本案例目的在于,通过一个具体的车间设备多行布局优化实例展示遗传算法如何解决上述问题。

7.5.2 数学模型

求解车间设备布局问题的目的是对车间内设备的位置进行布局,使物料搬运成本最低。为了简便起见,这里直接给出经过简化的数学模型。如图 7.6 所示。

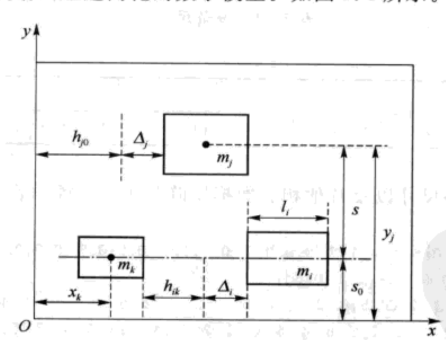


图 7.6 设备布局参数、决策变量和参考线

车间的多行设备布局是使设备间总的物料搬运成本最小化。其目标函数可表达为

$$C_{\min} = \sum_{i=1}^{n}\sum_{j=1}^{n} P_{ij} Q_{ij} D_{ij} \tag{7-25}$$

设备间距离表达式如下:

$$D_{ij}=|x_i-x_j|+|y_i-y_j| \tag{7-26}$$

设备横坐标求解公式：

$$x_i=x_k+(l_i+l_k)/2+h_{ik}+\Delta_i=h_{k0}+\Delta_k+(l_i+2l_k)/2+h_{ik}+\Delta_i \tag{7-27}$$

设备纵坐标求解公式：

$$y_i=(k-1)s+s_0\text{，如果 } z_{ik}=1, i=1,2,\cdots,n; k=1,2,\cdots,m \tag{7-28}$$

横坐标和纵坐标满足求同一行设备不重叠约束条件，即

$$|x_i-x_j|\geqslant[(l_i+l_j)/2+h_{ij}]z_{ik}z_{jk},\quad i,j=1,2,\cdots,n \tag{7-29}$$

其中：

$$z_{ik}=\begin{cases}1,\text{设备 } i \text{ 在第 } k \text{ 行上}\\0,\text{其他}\end{cases}\quad i=1,2,\cdots,n;\ k=1,2,\cdots,m \tag{7-30}$$

同时要求一台设备只能出现一次，即

$$\sum_{k=1}^{m}z_{ik}=1,\ i=1,2,\cdots,n\quad x_i,y_i\geqslant 0,\Delta_i\geqslant 0,i=1,2,\cdots,n. \tag{7-31}$$

上述模型中出现的符号的含义如下：

C 为搬运的总费用；n 为设备的总数量；m 为设备布局的总行数；l_i 为设备 i 的长度；h_{ij} 为设备 i 和 j 之间的横向最小间距要求；h_{i0} 为设备 i 和车间边界的横向最小间距要求；Δ_i 为设备 i 和设备 $i-1$ 或边界之间的净距离，取值范围为[0,1.5]；s 为设备行间距；s_0 为第一行设备到车间边界的距离；x_i 为设备 i 中心的 x 坐标；y_i 为设备 i 中心的 Y 坐标；P_{ij} 为设备 i 和 j 之间的单位物料每单位距离的搬运费用；Q_{ij} 为设备 i 和 j 之间物料搬运频率；D_{ij} 为设备 i 和 j 之间的矩形距离。

数据：假设共有10台设备（各设备尺寸见表7-3），在长×宽=10 m×8 m的车间内布局。

表7-3 设备尺寸

m

设备	1	2	3	4	5	6	7	8	9	10
尺寸	1.2×1.0	0.5×0.4	1.7×0.8	0.7×0.7	1.0×0.8	0.8×0.8	0.5×0.8	1.4×0.8	0.8×0.8	1.2×0.6

上述模型需要的设备尺寸以及其他相关数据矩阵如下（以M语言表示的矩阵形式给出）：

```
DeviceSize=[1.2 1.0;0.5 0.4;1.7 0.8;0.7 0.7;1.0 0.8;0.8 0.8;0.5 0.8;...
          1.4 0.8;0.8 0.8;1.2 0.6];
Pij=[0 8 2 1 6 4 3 2 3 5;8 0 3 2 6 7 3 4 2 5;2 3 0 3 4 2 4 3 1 6;
    1 2 3 0 4 3 2 7 2 2;6 6 4 4 0 5 2 3 4 1;4 7 2 3 5 0 5 6 3 2;
    3 3 4 2 2 5 0 4 4 4;2 4 3 7 3 6 4 0 3 2;3 2 1 2 4 3 4 3 0 2;...
    5 5 6 2 1 2 4 2 2 0];
Qij=[0 3 2 1 2 1 1 0 2 1;3 0 2 0 1 1 3 2 0 1;2 2 0 1 1 0 1 3 2 1;
    1 0 1 0 2 2 0 3 0 1;2 1 1 2 0 1 1 1 1 0;1 1 0 2 1 0 2 1 3 2;
    1 3 1 0 1 2 0 2 3 3;0 2 3 3 1 1 2 0 1 3;2 0 2 0 1 3 3 1 0 2;...
    1 1 1 1 0 2 3 3 2 0];
hi0=[1 2 1.5 1.5 1.3 1.2 1 1.8 2 1];
```

```
hij=[0 1 1.8 1.8 1.2 1.4 2 1.3 1.2 1;1 0 2 2 1 1.5 1.7 1.6 1.5 1;
    1.8 2 0 1 1 2 1 1.5 1.3 1;1.8 2 1 0 1 2 1.8 1.8 1 1;
    1.2 1 1 1 0 1.7 1 1.7 2 1;1.4 1.5 2 2 1.7 0 2 1 1 1;
    2 1.7 1 1.8 1 2 0 1.2 1 1;1.3 1.6 1.5 1.8 1.7 1 1.2 0 1.8 1;
    1.2 1.5 1.3 1 2 1 1 1.8 0 1;1 1 1 1 1 1 1 1 1 0];
s=2;
s0=1.5;
```

【补充说明】 读者可能对设备横坐标求解公式(7-28)有疑问,即为什么认为 x_k 就是和车间边界相邻,从而得到表达式 $x_k=h_{k0}+\Delta_k+\frac{l_k}{2}$? 其实这是为了简便起见,假设它与边界相邻,是对设备横坐标之间关系的一个说明。从后面遗传算法求解过程中染色体编码可以知道设备的排序,公式(7-28)可以被认为是一个递推公式,先确定第一行第一个设备的横坐标,然后按照上述规则以及自动换行规则依次推出其余设备的。

关于 h_{ij} 应当这样理解:不同的设备位于同一行时,其横向最小间距由 h_{ij} 矩阵给出,但是如果它们位于不同的行,横向间距就不限制了。也就是说,两设备只有在横向相邻时相应的 h_{ij} 才发挥作用。

设备和最右侧边界只有横向最小距离限制,并且等于最左侧边界横向最小距离的限制。从数值上讲,也就是对应每台设备的 h_{i0}。

7.5.3 算法步骤

本例采用遗传算法求解,有关基本的遗传算法在上一节已经做了介绍。这里只对每步的具体策略进行描述。

编码:编码表达采用设备符号和净间距两个列表的扩展换位表达方式:

$$[\{m_1,m_2,\cdots,m_n\},\{\Delta_1,\Delta_2,\cdots,\Delta_n\}]$$

其中,m_i 代表设备 i;Δ_i 代表设备 $i-1$ 和设备 i 之间的净间距。同时,采用自动换行策略,即如果在同一行内的各设备长度和相互实际间距之和超过了最大横向空间长度限制,本行最后一台设备自动进入下一行。

初始种群:随机产生。为了加快遗传算法的收敛过程,可以先用一个其他方法得到的比较优的设备符号序列来代替初始种群中第一个染色体设备符号序列。本例中可以先用(2 8 5 4 1 7 3 9 10 6)来代替。这样就形成了初始种群。

惩罚项:因为采用自动换行策略,在 X 方向上不会发生设备超出车间区域。因此,只要判断最后一行设备在 Y 轴方向上是否超出车间区域。

$$P_k=\begin{cases}0, & s_0+(m-1)s\leqslant H\\ T, & \text{其他}\end{cases}$$

其中,H 为车间的宽度;P_k 为不合理惩罚项;T 为正的大数惩罚值500。

适应度函数:对于染色体 v_k 的适应度函数定义为

$$fit(v_k)=1/(C_k+P_k) \tag{7-32}$$

其中,C 为目标函数。

选择:采用轮转法选择机制,即各个个体被选中的概率与其适应度大小成正比。设群体大

小为 M，个体 i 的适应度为 F_i，则个体 i 被选中的概率 $P_i = F_i / \sum_{i=1}^{m} F_i (i=1,2,\cdots,M)$。也就是说，种群按此概率进行选择，得到新的种群，适应度越高的个体被选中的概率越大。

交叉：用 PMX(部分映射交叉)方法处理设备排列序列交叉操作；用算术交叉方法处理净间距序列。

部分映射交叉：随机选取两个交叉点，两点内为需要交叉部分，然后按顺序先逐个进行交叉，同时修改交叉后的重叠部分，如图 7.7 所示。

部分映射交叉的操作过程如下：随机得到两竖线位置为交叉点，两线之间部分的所有元素需逐个交叉。首先交叉 H 和 D，交叉结束后将竖线外的重叠元素修改，如，第一行会出现两个 D，第二行会出现两个 H，然后进行修改，使每行元素不重复。以此继续交叉，最终得到新的个体 T'_x，T'_y。

T_x=(A D B | H F I | J G E C)
T_y=(B C A | D E J | H I F G)
→ (A H B | D F I | J G E C)
→ (B C A | H E J | D I F G)
→ (A H B | D E I | J G F C)
→ (B C A | H F J | D I E G)
→ (A H B | D E J | I G F C)=T'_x
→ (B C A | H F I | D J E G)=T'_y

图 7.7　部分交叉映射示意图

算术交叉：假如在两个个体 X_A^t，X_B^t 之间进行算术交叉，则交叉运算后产生的新个体是

$$X_A^{t+1} = aX_B^t + (1-a)X_A^t \tag{7-33}$$

$$X_B^{t+1} = aX_A^t + (1-a)X_B^t \tag{7-34}$$

其中，$a=0.5$。

变异：只对设备净间距进行变异操作，假设给定染色体的净间距序列是

$$\{\Delta_1, \Delta_2, \cdots, \Delta_i, \cdots, \Delta_n\}$$

依变异概率指定其变异点 Δ_i，r 是一个给定的整数，$[U_{\min}, U_{\max}]$ 是设备净间距的取值范围，则在该区间内可随机产生 r 个净间距：$\Delta_i^1, \Delta_i^2, \cdots, \Delta_i^r$，用它们分别取代变异点 Δ_i，可产生 r 个新染色体，从这 r 个新染色体中挑选最好的一个取代原来的染色体。

$$r=10,\ [U_{\min}, U_{\max}]=[0, 1.5]$$

运行参数：种群数量 50，终止代数为 200，交叉概率为 0.6，变异概率为 0.1。

7.5.4　求解代码

根据 7.5.3 描述的算法步骤可以写出如下的求解代码：

```
function  [chrx,MOC,PkG,LayoutG] = GA4WorkshopDeviceLayout
% chrx:存储每一代的最优解;MOC:存储每一代的最优解的目标函数值,
% PkG 存储每一代的最优解的适应度,
% LayoutG,存储每一代最优设备排列方式
DeviceSize = [1.2  1.0;0.5  0.4;1.7  0.8;0.7  0.7;1.0  0.8;0.8  0.8;...
    0.5  0.8;1.4  0.8;0.8  0.8;1.2  0.6];
Pij = [0  8  2  1  6  4  3  2  3  5;8  0  3  2  6  7  3  4  2  5;2  3  0  3  4  2  4  3  1  6;
     1  2  3  0  4  3  2  7  2  2;6  6  4  4  0  5  2  3  4  1;4  7  2  3  5  0  5  6  3  2;
     3  3  4  2  2  5  0  4  4  4;2  4  3  7  3  6  4  0  3  2;3  2  1  2  4  3  4  3  0  2;...
     5  5  6  2  1  2  4  2  2  0];
```

```
Qij=[0 3 2 1 2 1 1 0 2 1;3 0 2 0 1 1 3 2 0 1;2 2 0 1 1 0 1 3 2 1;
     1 0 1 0 2 2 0 3 0 1;2 1 1 2 0 1 1 1 1 0;1 1 0 2 1 0 2 1 3 2;
     1 3 1 0 1 2 0 2 3 3;0 2 3 3 1 1 2 0  1 3;2 0 2 0 1 3 3 1 0 2;...
     1 1 1 1 0 2 3 3 2 0];
hi0=[1 2 1.5 1.5 1.3 1.2 1 1.8 2 1];
hij=[0  1 1.8 1.8 1.2 1.4 2 1.3 1.2 1;1 0 2 2 1 1.5 1.7 1.6 1.5 1;
     1.8 2 0 1 1 2 1 1.5 1.3 1;1.8 2 1 0 1 2 1.8 1.8 1 1;
     1.2 1 1 1 0 1.7 1 1.7 2 1;1.4 1.5 2 2 1.7 0 2 1 1 1;
     2 1.7 1 1.8 1 2 0 1.2 1 1;1.3 1.6 1.5 1.8 1.7 1 1.2 0 1.8 1;
     1.2 1.5 1.3 1 2 1 1 1.8 0 1;1 1 1 1 1 1 1 1 1 0];
s=2;
s0=1.5;
nchr=50; %种群数量
G=200; %代终止数
pc=0.6; %交叉概率
pm=0.1; %变异概率
T=500;
H=8; %车间宽
r=10;
Umin=0;
Umax=1.5;
%% 计算距离矩阵的函数(采用 nested function 结构,以 function + end 标识,位置可以出
%现在主函数体的任何位置)
%    chr:染色体。Dij:返回的距离矩阵
    function [Dij,nRow,Layout]=GenerateDistanceMatrix(chr)
        m=chr(1:10);
        delta=chr(11:20);
        %记录每行排列的设备号,由于行数不定,但最多不可能超过 10 行,所以用一个 10*1 的
        %cell 数组记录。
        Layout=cell(10,1);
        k=1; %染色体设备序列 m 中第一个设备号
        nRow=1; %排列的行数
        while k<10  %while 循环:按照自动换行策略判断设备是否排列完
            for kk=k:10
                if sum(DeviceSize(k:kk,1))+sum(delta(k:kk))+...
                        sum( hij( sub2ind( size(hij), m(k:kk-1), m(k+1:kk) )))...
                        +sum( hi0( m([k,kk]) ) )>10
                    %判断是否该换行,第一项:设备总长,第二项:净间距和,第三项和第
                    %四项相应的 hij 和 hi0 之和
                    Layout{nRow}=m(k:kk-1);
                    k=kk;
                    break;
                end
            end
            nRow=nRow+1;
            %判断是否到最后一行,并布置最后一行的设备
            if kk==10
                Layout{nRow}=m(kk);
            elseif sum(DeviceSize(kk:end,1))+sum(delta(kk:end))+...
                sum( hij( sub2ind( size(hij), m(kk:end-1), m(kk+1:end) )))...
```

```
                + sum( hi0( m([kk,end]) ) ) <= 10
            Layout{nRow} = m(kk:end);
            break
        end
    end
    x = zeros(10,1); % 存储各设备 x 坐标
    y = zeros(10,1); % 存储各设备 y 坐标
    for nr = 1:nRow
        x( Layout{nr}(1) ) = hi0( Layout{nr}(1) ) + delta( Layout{nr}(1) );
        y(Layout{nr}) = (nr - 1) * s + s0;
        if length(Layout{nr}) == 1
            % 某行只有一个设备,上面已经得到设备的横纵坐标,直接循环下一个 nr
            continue;
        end
        for nc = 2:length(Layout{nr})
            x(Layout{nr}(nc)) = x(Layout{nr}(nc - 1)) + ...
                mean( DeviceSize(Layout{nr}([nc - 1,nc]),1) ) + ...
                hij( Layout{nr}(nc - 1),Layout{nr}(nc) ) + delta(Layout{nr}(nc));
        end
    end
    Dij = zeros(10);
    for ii = 1:10
        for jj = 1:10
            Dij(ii,jj) = abs(x(ii) - x(jj)) + abs( y(ii) - y(jj) );
        end
    end
end
%% 随机生成 nchr 个初始种群,用已有设备序列替换掉前三个染色体的设备序列段
% M,由于是对随机数矩阵每行排序后新矩阵中的数字在原矩阵的序号,所以 M 相当于随机生成了
% nchr 个 1 到 10 的随机排序
[sorted,M] = sort(rand(nchr,10),2);
CHR = [M,unifrnd(Umin,Umax,nchr,10)] ;
for ki = 1:3
    % 用所给的设备序列替换掉前三个染色体的设备序列段
    CHR(ki,1:10) = [2 8 5 4 1 7 3 9 10 6];
end
%% 目标函数以及适应度函数,设备排列,根据适应度函数特点,将目标函数和适应度函数
% 以及设备排列用一个函数实现,减少重复计算
    function [C,Pk,Layout] = ObjfunEvalfun(chr)
        [Dij,nRow,Layout] = GenerateDistanceMatrix(chr);
        C = sum( sum(Pij. * Qij. * Dij) ); % 目标函数值
        Pk = 1/( C + T * ( (nRow - 1) * s + s0 > H ) ); % 适应度函数值
    end

%% 进化 G 次,PkG 存储每一代的最优解的适应度,MOC,存储每一代的最优解的目标函数值,
% chrx 存储每一代的最优解,C,进化过程中每代染色体的目标函数值,Pk,进化过程中每代染色体
% 的适应度,LayoutG,存储每一代最优设备排列方式,Layoutnchr,每一代中种群各个染色体的
% 设备排列方式
PkG = zeros(G,1);
```

```
MOC = zeros(G,1);
chrx = zeros(G,20);
LayoutG = cell(G,1);
Layoutnchr = cell(nchr,1);
C = zeros(nchr,1);
Pk = zeros(nchr,1);
for g = 1:G
    %% 计算目标函数值以及适应度
    for kn = 1:nchr
        [C(kn),Pk(kn),Layoutnchr{kn}] = ObjfunEvalfun(CHR(kn,:));
    end
    [sortPk, ind] = sort(Pk);
% 进化的代数大于 1,但是最大适应度值不如上一代的优。用上一代的最优结果替代本代最优结果.
    if g > 1 && sortPk(end) < PkG(g - 1)
        PkG(g) = PkG(g - 1);
        chrx(g,:) = chrx(g - 1,:);
        MOC(g) = MOC(g - 1);
        LayoutG{g} = LayoutG{g - 1};
    else
        PkG(g) = sortPk(end);
        chrx(g,:) = CHR(ind(end),:);
        MOC(g) = C(ind(end));
        LayoutG{g} = Layoutnchr{ind(end)};
    end
    %% 选择操作
    SelectP = Pk/sum(Pk); % 选择概率
    % 01 区间划分(各区间长度依次等于各个体选择的概率比例)
    Interval01Divide = cumsum(SelectP);
    Test = rand(nchr,1); % 生成 nchr(50)个 0,1 随机数
    % 随机数落入的区间序号,表示该区间代表的个体被选中
    TestResult = arrayfun(@(x) find(Interval01Divide > = x,1,'first'),Test);
    Frequency = accumarray(TestResult,1); % 各个体选中的次数,进而决定各个体复制的次数
    Chr = cell(length(Frequency),1);
    for tt = 1:length(Frequency)
        % 根据 Frequency 每个分量决定相应个体的复制次数
        Chr{tt} = repmat(CHR(tt,:),Frequency(tt),1);
    end
    Chr = cell2mat(Chr);
    %% 该部分是交叉操作
    RandMatch = randperm(nchr); % 随机配对
    for tt = 1:nchr/2
        if rand > 1 - pc % 满足交叉概率,下面交叉
            a1 = Chr(RandMatch(2 * tt - 1),1:10); % a1,a2 进行交叉的染色体设备序列段
            a2 = Chr(RandMatch(2 * tt),1:10);
            b1 = Chr(RandMatch(2 * tt - 1),11:20); % b1,b2 进行交叉的染色体 delta 段
            b2 = Chr(RandMatch(2 * tt),11:20);
            Points = sort(unidrnd(10,1,2));
            % 交叉 a1,a2
            for uu = 0:range(Points)
                a1(a1 = = a2(Points(1) + uu)) = a1(Points(1) + uu);
```

```
                temp = a1(Points(1) + uu);
                a1(Points(1) + uu) = a2(Points(1) + uu);
                a2(a2 == temp) = a2(Points(1) + uu);
                a2(Points(1) + uu) = temp;
            end
            %交叉 b1,b2
            alpha = unidrnd(2,1,10) - 1;
            temp2 = b1. * alpha + b2. * (1 - alpha);
            b2 = b1. * (1 - alpha) + b2. * alpha;
            b1 = temp2;
            Chr(RandMatch(2 * tt - 1),:) = [a1,b1];
            Chr(RandMatch(2 * tt),:) = [a2,b2];
        end
    end
    %% 该部分是变异操作
    for tt = 1:nchr
        if rand < pm %满足变异概率
            pos = unidrnd(10,1) + 10; %生成 delta 变异位置(10个位置中的随机一个)
            delta_r = unifrnd(Umin,Umax,1,r); %随机生成 r 个 delta
            PkTemp = zeros(1,r);
            for tr = 1:r
                Chr(tt,pos) = delta_r(tr);
                [Ctemp,PkTemp(tr)] = ObjfunEvalfun(Chr(tt,:));
            end
            [maxPktemp,indmax] = max(PkTemp);
            Chr(tt,pos) = delta_r(indmax);
        end
    end
%% 种群更新
    CHR = Chr;
end
plot(1:G,MOC);
xlabel('进化次数');
ylabel('最优目标函数值');
title('进化次数同最优函数值曲线')
end
```

多运行几遍上述程序，可以得到一些比较优的解。由于遗传算法的随机性，每次运行的结果可能不一样。但得到的解是比较优的。比如下面的一组解：

$[\{m_1^*, m_2^*, \cdots, m_n^*\}, \{\Delta_1^*, \Delta_2^*, \cdots, \Delta_n^*\}]$=[5, 2, 1, 8, 6, 10, 3, 4, 9, 7, 0.076 0, 1.080 2, 0.038 1, 0.014 5, 0.153 2, 0.064 8, 0.086 9, 0.004 8, 0.121 4, 0.127 9]

对应的目标函数值为1957。按照自动换行策略，上述得到的最优设备排列方式为

第一行：5　2　1

第二行：8　6　10　3

第三行：4　9　7

整个遗传算法进化次数和最优目标函数值的关系如图7.8所示，每次运行的结果图形会不一样。

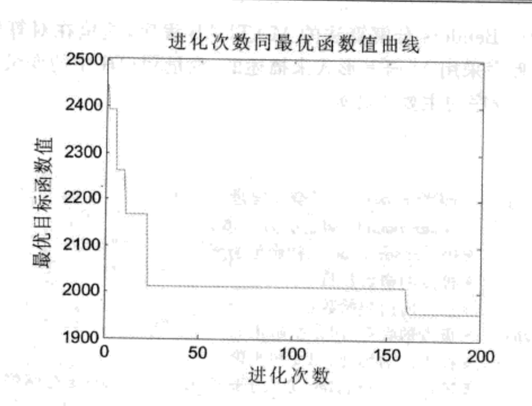


图 7.8 进化曲线图

从图 7.8 可以看出，随着进化次数的增加，最优的目标函数值也在不断减小，遗传算法最终得到了一个优化的解。感兴趣的读者可以多运行几遍上述程序，观察一下都可以产生哪些优化的解。

7.6 案例 14：应用 Benders 分解算法求解混合 0－1 规划

7.6.1 概 述

MATLAB 的优化工具箱对于整数规划一直支持力度不够，目前仅有 bintprog 函数来求解纯的 0－1 规划问题，对于混合规划一直没有现成的函数可以求解，只能借助第三方工具箱。我们知道，对于任意非负整数 z，可以将其表示成 $z=\sum_{i=0}^{n}2^i x_i$ ($x_i\in\{0,1\}$)的形式。如果是负整数，则前面乘以－1 即可。因此混合整数规划也可以转化为混合 0－1 规划。

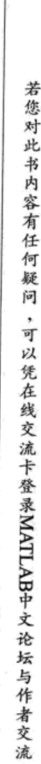
当然，直接求解整数规划问题的方法也有，如分支定界法和割平面法等。但在求解大型问题时，上述方法往往很难奏效。本案例主要介绍如何利用 Benders 分解算法来求解混合 0－1 规划问题。Benders 分解算法是根据对偶理论得到的求解混合整数规划的有效算法，关于其理论介绍，有兴趣的读者可以参考文献[16]进行了解。这里为简洁起见，只给出算法的基本描述与步骤。

7.6.2 Benders 分解算法

考虑如下优化问题：

$$\begin{aligned} &\min \quad \boldsymbol{C}x+\boldsymbol{D}y \\ &\text{s.t.} \quad \boldsymbol{A}x+\boldsymbol{B}y\geqslant \boldsymbol{b} \\ &x\in\{0,1\},y\geqslant 0 \end{aligned} \tag{7-35}$$

其中，$\boldsymbol{A}$ 为 nRowA * nColA 的矩阵；$\boldsymbol{B}$ 为 nRowB * nColB 的矩阵；$\boldsymbol{C},\boldsymbol{D},\boldsymbol{b}$ 分别是长为 nColA，nColB，nRowA(nRowB) 的行(列)向量。



为了方便读者阅读 Benders 分解算法的 MATLAB 程序，这里在对算法描述时是结合具体 Benders 分解算法程序采用 M 语言形式来描述的，变量和程序中的变量是统一的。

Benders 分解算法程序的主要步骤如下：

(1) 初始化

```
x0 = zeros(nColA,1);        % x0 为 nColA * 1 的全 0 向量
LB = - 1e10; LB;            % lower bound，初始值为 - 10^10
UB = inf;                   % UB，upper bound，初始值为无穷
p = 0;                      % 极点的初始数目
q = 0;                      % 极方向的初始数目
U = zeros(100,nRowA);       % 极点的集合，用 0 初始化
V = zeros(100,nRowA);       % 极方向的集合，用 0 初始化
epsilon = 0.99;             % LB/UB 的阈值，LB/UB 大于这个阈值时，benders 分解停止
```

(2) 求解子问题

求解如下子问题：

$$\begin{aligned} \min \quad & (\boldsymbol{A}x_0-\boldsymbol{b})^{\mathrm{T}}\boldsymbol{UorV} \\ \text{s.t.} \quad & \boldsymbol{B}^{\mathrm{T}}\boldsymbol{UorV}\leqslant \boldsymbol{D}^{\mathrm{T}} \end{aligned} \tag{7-36}$$

程序中相应代码：

```
[UorV,fval,exitflag] = linprog((A * x0 - b),B',D',[],[],zeros(nRowB,1),...
inf(nRowB,1),[],options);
```

根据 exitflag 判断，如果 exitflag=1，则求得一极值点 UorV，那么

```
p = p + 1;
U(p,:) = UorV';  % U 矩阵的第 p 行赋值 UorV'
```

并令：

```
UB = C * x0 + UorV' * (b - A * x0);
```

如果 exitflag=－3，则原问题无界，可以求得一个极方向 UorV，那么

```
q = q + 1;
V(q,:) = UorV';  % V 矩阵的第 q 行赋值 UorV'
```

其他情况，无解，退出。

(3) 求解主问题

求解如下主问题：

$$\begin{aligned} & \min \quad z \\ \text{s.t.} \quad & z > \boldsymbol{C}x_0+\boldsymbol{U}(h,:)(\boldsymbol{b}-\boldsymbol{A}x), \quad h=1,2,\cdots,p \\ & \boldsymbol{V}(s,:)(\boldsymbol{b}-\boldsymbol{A}x)\leqslant 0, \quad s=1,2,\cdots,q \end{aligned} \tag{7-37}$$

如果 UB 为无穷(inf)，令 $k=17$(为了表示 z 而对 i 设的一个较大的上限)；否则 $k=$ceil

(log2(UB+1))−1;其中 ceil( )表示向上取整;将 z 表示成

$$z=\sum_{i=0}^{k}2^{i}zx_{i},\quad zx_{i}\in\{0,1\} \tag{7-38}$$

zx_i 和式(7-37)中 x 都是 0-1 变量,合在一起用 ZX 来表示,其中,ZX 前$(k+1)$个分量是 zx_i,从$(k+2)$个开始到末尾是 x,这也就是为什么后面程序中得到 OptZX 后,令

```
x0 = OptZX(k + 2:end);
```

再令

```
CoefZ = 2.^(0:k);  % zx 的系数向量
    f = [CoefZ';zeros(nColA,1)]; % 主问题目标函数的系数向量
CoefMatZX = [repmat( - CoefZ,p,1), - U(1:p,:) * A;zeros(q,k + 1), - V(1:q,:) * A];
bZX = [ - (repmat(C * x0,p,1) + U(1:p,:) * b); - V(1:q,:) * b];
```

那么式(7-37)可以改写成

$$\begin{aligned}&\min\quad \boldsymbol{f}^{\mathrm{T}}\boldsymbol{ZX}\\&\text{s.t.}\quad \boldsymbol{CoefMatZX}*\boldsymbol{ZX}\leqslant\boldsymbol{bZX}\end{aligned} \tag{7-39}$$

问题式(7-39)可用分支定界法求解,当式(7-40)规模较大时,还可以用一些智能优化算法,如遗传算法、模拟退火算法等求解。这里仅介绍利用 MATLAB 中的函数 bintprog 求解的方法。

求解式(7-39)的相应的代码如下:

```
% bintprog,分支定界法
[OptZX,minZ,ExitflagBint] = bintprog(f,CoefMatZX,bZX,[],[],[],OptionsBint);
```

得到 OptZX 后,令

```
x0 = OptZX(k + 2:end);
```

比较 LB/UB 的值,如果小于 epsilon,算法继续分解,并转第二步;否则 OptX = x0 = OptZX(k+2:end);是最优解,停止分解,将 OptX 带入原问题,求解如下问题得到最终的原始问题的解:

$$\begin{aligned}&\min\quad \boldsymbol{DY}\\&\text{s.t.}\quad -\boldsymbol{BY}\leqslant\boldsymbol{A}*\boldsymbol{OptX}-\boldsymbol{b}\end{aligned} \tag{7-40}$$

相应代码如下:

```
[OptY,OptValue] = linprog(D', - B,A * OptX - b,[],[],zeros(nColB,1),...
inf(nColB,1),[],options);
```

下面是完整的 Benders 分解算法程序代码以及相应的流程图:

```
function [OptX,OptY,OptValue] = BendersDecomposition(C,D,A,B,b)
%  应用 Benders 分解算法解决混合整数规划问题
%  调用格式:[OptX,OptY,OptValue] = BendersDecomposition(C,D,A,B,b)
%  用来解决如下形式的规划问题:
```

```
%          min  C*x+D*y
%          s.t. A*x+B*y>=b; x,0-1变量,y>=0

%迭代停止的判断阈值:LB/UB大于epsilon时停止迭代。可以根据问题复杂程度灵活调节
epsilon=0.99;
[nRowA,nColA]=size(A);
[nRowB,nColB]=size(B);
%step 1:初值化
x0=zeros(nColA,1); %初始值
LB=-1e10;
UB=inf;
p=0;
q=0;
U=zeros(100,nRowA);
V=zeros(100,nRowA);
options=optimset('LargeScale','off','Simplex','on');
OptionsBint=optimset('MaxRLPIter',100000,'NodeSearchStrategy','bn',...
    'MaxTime',50000);
while LB/UB<epsilon
    %step 2:求解子问题
    [UorV,fval,exitflag]=linprog((A*x0-b),B',D',[],[],zeros(nRowB,1),...
        inf(nRowB,1),[],options);
    if exitflag==1 %得到极值点
        p=p+1;
        U(p,:)=UorV'; %极值点
        UB=C*x0+U(p,:)*(b-A*x0);
    elseif exitflag==-3 %无界
        q=q+1;
        V(q,:)=UorV'/1e16; %极方向
        V(q,V(q,:)<0.0001)=0;
    else
        msgbox('无可行解!')
        return;
    end
    %step 3:求解主问题
    if isinf(UB)
        k=17;
    else
        k=ceil(log2(UB+1))-1; %根据2^(k+1)-1>UB决定最小的k值
    end
    CoefZ=2.^(0:k); %z0到zk的系数向量
    f=[CoefZ';zeros(nColA,1)]; %主问题目标函数的系数向量
    CoefMatZX=[repmat(-CoefZ,p,1),-U(1:p,:)*A;zeros(q,k+1),-V(1:q,:)*A];
    bZX=[-(repmat(C*x0,p,1)+U(1:p,:)*b);-V(1:q,:)*b];
    [OptZX,minZ,ExitflagBint]=bintprog(f,CoefMatZX,bZX,[],[],[],...
        OptionsBint);
    if ExitflagBint==1
```

```
        LB = minZ;
        x0 = OptZX(k + 2:end);
    else
        break;
    end
    LB,UB % #ok < NOPRT >
end
OptX = x0;
options = optimset('LargeScale','off','Simplex','on');
[OptY,OptValue] = linprog(D', - B,A * OptX - b,[],[],zeros(nColB,1),inf(nColB,1),...
    [],options);
OptValue = C * OptX + D * OptY;
```

Bender 分解算法流程如图 7.9 所示。

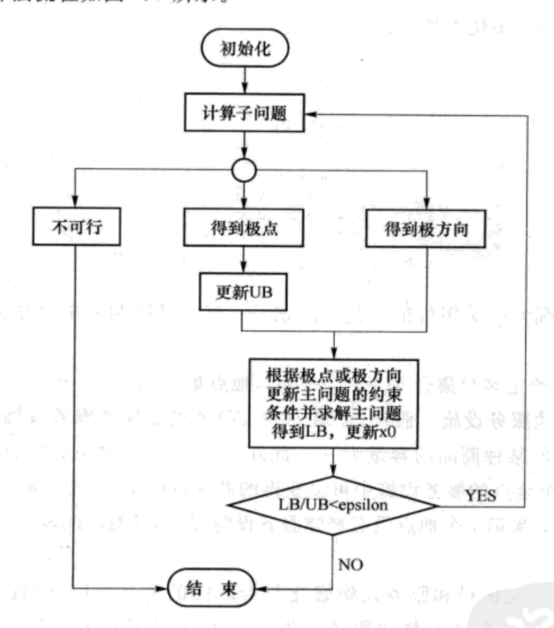


图 7.9 Benders 分解算法流程图

7.6.3 实例分析

【例 7.6-1】 求如下混合 0-1 规划的解。

$$\min \quad 5x_1-6x_2+7y_1+3y_2$$
$$\text{s.t.} \quad y_1-2x_1+3x_2\geqslant 0$$
$$y_1+y_2+2x_1-x_2\geqslant 11$$
$$x_1+x_2+y_2\geqslant 7$$

$$y_1, y_2 > 0$$
$$x \in \{0,1\}$$

将上式写成式(7-36)的标准形式得到：

$\boldsymbol{C}=[5,-6]$，$\boldsymbol{D}=[7,3]$，$\boldsymbol{A}=[-2,3;2,-1;1,1]^{\mathrm{T}}$，$\boldsymbol{B}=[1,0;1,1;0,1]$，$\boldsymbol{b}=[0,11,7]^{\mathrm{T}}$

利用 BendersDecomposition 函数求解的代码如下：

```
>> C = [5 - 6];
D = [7,3];
A = [-2 3;2, -1;1 1];
B = [1 0;1 1;0 1];
b = [0;11;7];
[OptX,OptY,OptValue] = BendersDecomposition(C,D,A,B,b);
```

得到的最优解以及最优值如下：

```
OptX =
     1
     1
OptY =
     0
    10
OptValue =
    29
```

再看一个有很强实际应用价值的优化问题——有容量限制的单商品货物流的设施选址问题。

【例 7.6-2】 给定客户集合 $M=\{1,\cdots,m\}$，地点集合 $N=\{1,\cdots,n\}$，希望从地点集合 N 中选择若干地点修建服务设施。假设，在第 $j(j\in N)$ 个地点修建服务设施的费用为 f_j，该服务设施所能够存储的某种商品的容量为 u_j。此外，第 i 个客户对该商品的需求量为 b_i，令 x_{ij} 表示客户 i 从第 j 个地点的服务设施中可以获得的商品供应量，c_{ij} 表示客户 i 到地点 j 获取单位商品时的成本。如果第 j 个地点没有修建服务设施，就不可能由此地向客户供应商品，提供服务。

为了描述商品的运输量和服务设施修建与否之间的关系，可以引进 0-1 变量 $y\in B^n$，其中，$y_j=1$ 表示在第 j 个地点修建服务设施，$y_j=0$ 表示不在第 j 个地点修建服务设施。于是，有容量限制的单商品货物流的设施选址问题可以用下面的线性混合 0-1 规划模型来描述：

$$\begin{aligned} \min \quad & \sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij} + \sum_{j\in N} f_j y_j \\ \text{s. t.} \quad & \sum_{j\in N} x_{ij} = b_i, \quad i\in M, \\ & \sum_{i\in M} x_{ij} - u_j y_j \leqslant 0, \quad j\in N, \\ & x_{ij}\geqslant 0, \quad i\in M, j\in N, y\in B^n \end{aligned} \tag{7-42}$$

下面结合具体数据来求解上面的模型。假设客户集合 $M=\{1,2,\cdots,10\}$，地点集合为 $N=\{1,2,\cdots,6\}$。其他一些必要数据见表 7-4～表 7-7。

表 7-4　f_j 数据

地点	1	2	3	4	5	6
费用/万元	110	150	130	170	140	120

表 7-5　u_j 数据

地点	1	2	3	4	5	6
容量/t	1000	1300	1200	1600	1400	1100

表 7-6　b_i 数据

客户	1	2	3	4	5	6	7	8	9	10
需求量/t	300	400	350	430	370	500	280	460	380	430

表 7-7　c_{ij} 数据(i:列，j:行)

77	66	80	76	74	75	74	76	75	63
79	71	70	80	76	61	67	76	76	70
63	80	77	74	75	66	80	64	66	80
79	80	63	61	68	61	61	70	74	67
73	64	69	77	74	62	69	69	74	72
62	80	79	79	64	77	68	73	64	65

针对上述实际数据，相应的求解代码如下：

```
fj=[110  150  130  170  140  120]; % fj数据
uj=[1000  1300  1200  1600  1400  1100]; % uj数据
bi=[300  400  350  430  370  500  280  460  380  430]; % bi的数据
cij=[77  66  80  76  74  75  74  76  75  63
79  71  70  80  76  61  67  76  76  70
63  80  77  74  75  66  80  64  66  80
79  80  63  61  68  61  61  70  74  67
73  64  69  77  74  62  69  69  74  72
62  80  79  79  64  77  68  73  64  65
]; % cij数据，i对应列，j对应行
C = fj; % 标准形式中的C
D = cij(:)'; % 标准形式中的D
A = [zeros(20,6);diag(uj)]; % 标准形式中的A
% B1,B2,B3合起来为标准形式中的B，因为有等式约束，故将其写成> = b和< = b，这样等价于等式
% 约束，对应两个系数矩阵B1,B2,B3对应着sigma(xij) - uj * yj< = 0这个约束中的xij系数矩阵
B1 = zeros(10,60);
B2 = zeros(10,60);
B3 = zeros(6,60);
for k = 0:9
  B1(k + 1,6 * k + 1:6 * k + 6) = ones(1,6);
```

```
    B2(k + 1,6 * k + 1:6 * k + 6) =- ones(1,6);
end
ind = 1:6:55; % B3 第一行不为 0 的系数对应的 x 变量号
for k = 0:5
    B3(k + 1,ind + k) =- ones(1,10);
end
B = [B1;B2;B3];
b = [bi'; - bi';zeros(6,1)];
[OptX,OptY,OptValue] = BendersDecomposition(C,D,A,B,b);
```

运行上述代码得到的结果如下：

```
Exiting: The problem is unbounded; the constraints are not restrictive enough.
Optimization terminated.
LB =
     0
UB =
   Inf
Exiting: The problem is unbounded; the constraints are not restrictive enough.
Optimization terminated.
LB =
     0
UB =
   Inf
Exiting: The problem is unbounded; the constraints are not restrictive enough.
Optimization terminated.
LB =
     0
UB =
   Inf
Optimization terminated.
Optimization terminated.
LB =
     223232
UB =
     258710
Optimization terminated.
Optimization terminated.
LB =
     245260
UB =
     245260
Optimization terminated.
```

从上面结果可以看出，前三次迭代找到的都是极方向，第四次、第五次迭代找到的是极值点，从而完成整体的计算。找到的最优解见表7－8和表7－9。



表 7-8　x_{ij}^*

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	500	280	0	0	70
300	0	0	430	0	0	0	460	10	0
0	0	0	0	0	0	0	0	0	0
0	400	350	0	0	0	0	0	0	0
0	0	0	0	370	0	0	0	370	360

表 7-9　y_j^* 数据

地　点	1	2	3	4	5	6
修建与否	否	修	修	否	修	修

得到的最优目标函数值(最少费用)为:256 030 万元。

第 8 章

案例 15：人脸图像压缩与重建

8.1 概述

本案例的目的是介绍用统计方法(PCA 以及近些年发展起来的一些 PCA 的增强模型，如二维化模型 2DPCA、MatPCA 和模块化方法 Module PCA 等)进行人脸图像压缩与重建原理。为了方便读者试运行以及研究本案例中的程序，本案例算法选用的数据库为标准人脸数据库 ORL。

PCA (Principle Component Analysis) 是统计学中的主成分分析方法。主成分分析方法从矩阵角度讲也称 K-L 变换。这种方法应用于图像压缩时，首先将图像训练库里的每个二维图像拉伸成向量，然后对其进行主成分分析得到主成分的变换矩阵以及图像均值向量。图像压缩过程就是把待压缩的图像减去训练得到的图像均值向量并通过变换矩阵变换成维数很小的一个向量的过程。图像的重建就是将压缩的图像通过变换矩阵的逆变换后再加上图像均值向量得到的压缩前向量的近似向量。因为是主成分分析，所以图像会有较少的信息损失，并且不能完全复原，但是这种信息损失一般是非常小的。

以上是最基本的主成分分析方法对图像进行压缩和重建的大致过程。基本的 PCA 存在以下一些缺点：当人脸图像光照以及位置发生较大变化时，基本的 PCA 无法有效捕捉这些变化，而且有研究表明，基本的 PCA 几乎不能捕捉到图像之间的一些最简单的一致性，除非这些信息包含在训练图像中。除此之外，基本的 PCA 都要把图像的像素按某种方式(一般是各列首位相接)拉伸成一个维数很高的向量。当图像尺寸稍大时，这个拉伸后的向量维数会非常高，更不用说训练图像之间的协方差矩阵了。虽然利用 SVD 分解可以近似得到特像向量(特征图像)，从而避免生成巨大的协方差矩阵，但这样做很多时候是不精确的。

鉴于上述 PCA 方法的不足，人们提出了一系列的改进方法，即上面提到的 2DPCA、MatPCA 以及 Module PCA 等，下面将一一介绍这些方法。为方便起见，算法中讨论的图像均为标准人脸数据库 ORL 中的图像。

本案例还有一个目的就是通过将上述算法集成到 GUI 程序中，使读者能了解一些简单的 GUI 要素实现方法，如菜单、回调函数、不同控件之间的参数传递等。

8.2 基本的 PCA 方法实现人脸图像压缩与重建

8.2.1 K-L 变换

基本的 PCA 方法在对人脸图像压缩前首先要挑选一些图像作为训练图像。假设要训练的图像的尺寸为 $N\times N$，那么它的所有列的像素可以首尾相接串起来。这样，每一幅图像可以

拉伸成一个长为 N^2 的列向量，它可以被看成 N^2 维空间中的一个点。由于所训练的图像彼此之间都具有很多相似的地方，所以这些向量在这一超高维空间中的分布并不是随机或杂乱无章的，它们彼此之间具有很强的相关性，可以对其进行主成分分析（K－L 变换），即用一个低维子空间描述这些图像，同时又不丢失关键信息。

假设选定的训练图像集含有 M 幅图像，令 $\boldsymbol{x}_i, i\in\{1,2,\cdots,M\}$ 为第 i 个训练样本的图像向量，$\boldsymbol{x}=[\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_M]$，$\boldsymbol{u}$ 为所有训练样本图像的平均图像向量，即

$$\boldsymbol{u}=\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{x}_i$$

$$\boldsymbol{u}=[\boldsymbol{u},\boldsymbol{u},\cdots,\boldsymbol{u}]$$

主成分分析需要训练样本集的总体散布矩阵，即协方差矩阵：

$$\boldsymbol{\Sigma}=\mathrm{E}[(\boldsymbol{x}-\boldsymbol{u})(\boldsymbol{x}-\boldsymbol{u})^{\mathrm{T}}]$$

或

$$\boldsymbol{\Sigma}=\frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i-\boldsymbol{u})(\boldsymbol{x}_i-\boldsymbol{u})^{\mathrm{T}}$$

$\boldsymbol{\Sigma}$ 是一个维数为 $N^2\times N^2$ 的矩阵，主成分分析方法需要计算其特征值和正交归一的特征向量。由于实际应用中 N^2 都会非常大，直接计算是非常困难的。为此，下面的定理可以较好地解决这一问题。

定理 8.1(SVD) 设 $\boldsymbol{A}$ 是一个秩为 r 的 $n\times r$ 维矩阵，则存在两个正交矩阵：

$$\boldsymbol{U}=(\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r)\in\mathscr{R}^{n\times r}\quad \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{I};$$

$$\boldsymbol{V}=(\boldsymbol{v}_1,\boldsymbol{v}_2,\cdots,\boldsymbol{v}_r)\in\mathscr{R}^{\times r}\quad \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{I};$$

以及对角阵：

$$\boldsymbol{\Lambda}=\mathrm{diag}(\lambda_1,\lambda_2,\cdots,\lambda_r)\in\mathscr{R}^{\times r}\quad \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{I},$$

且

$$\lambda_1\geqslant\lambda_2\geqslant\cdots\geqslant\lambda_r$$

满足

$$\boldsymbol{A}=\boldsymbol{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{V}^{\mathrm{T}}\tag{8-1}$$

其中，$\lambda_i(i=1,2,\cdots,r)$ 为矩阵 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ 和 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 的非零特征值；$\boldsymbol{u}_i$ 和 $\boldsymbol{v}_i$ 分别为 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ 和 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 对应于 λ_i 的特征向量。上述分解成为矩阵 $\boldsymbol{A}$ 的奇异值分解（Singular Value Decomposition，SVD），$\boldsymbol{A}$ 的奇异值为 $\sqrt{\lambda_i}$。

推论

$$\boldsymbol{U}=\boldsymbol{A}\boldsymbol{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}\tag{8-2}$$

回到我们的问题，由于 $\boldsymbol{\Sigma}$ 可表示为

$$\boldsymbol{\Sigma}=\frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{x}_i-\boldsymbol{u})(\boldsymbol{x}_i-\boldsymbol{u})^{\mathrm{T}}=\frac{1}{M}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$$

其中：

$$\boldsymbol{X}=(\boldsymbol{x}_1-\boldsymbol{u},\boldsymbol{x}_2-\boldsymbol{u},\cdots,\boldsymbol{x}_M-\boldsymbol{u})$$

故，构造矩阵

$$\boldsymbol{R}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\in\mathscr{R}^{M\times M}$$

容易求出其特征值 λ_i 及相应的正交归一特征向量 $\boldsymbol{v}_i(i=1,2,\cdots,M)$。从上述推论可知，$\boldsymbol{\Sigma}$ 的正交归一特征向量为

$$\boldsymbol{u}_i=\frac{1}{\sqrt{\lambda_i}}\boldsymbol{X}\boldsymbol{v}_i,\quad i=1,2,\cdots,M\tag{8-3}$$

这就是图像的特征向量。它是通过计算较低维矩阵 $\boldsymbol{R}$ 的特征值和特征向量而间接求出的。

将特征值按照从大到小的顺序排列：$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_M$，其对应的特征向量为 $\boldsymbol{u}_i$。这样，每一幅人脸图像都可以投影到由 $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_M$ 张成的子空间中。因此，每一幅人脸图像对应于子空间中的一个点。同样，子空间中的任一点也对应于一幅图像。有了这样一个由 $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_M$ 张成的子空间，任何一幅人脸图像都可以向其投影并获得一组坐标系数，这组系数表明了该图像在子空间中的位置。换句话说，任何一幅人脸图像都可以表示为 $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_M$ 的线性组合，其加权系数即是 K－L 变换的展开系数，也可以称为该图像的代数特征。

对于任一待压缩的人脸图像 $\boldsymbol{f}$，可以通过向特征子空间投影得到其系数向量：

$$\boldsymbol{y}=\boldsymbol{U}^{\mathrm{T}}(\boldsymbol{f}-\boldsymbol{u}), \quad \boldsymbol{U}=(\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_M)$$

得到的系数向量 $\boldsymbol{y}$ 可以被认为是对它的压缩，因为这个系数向量维度 M 通常远远小于 f，从而可大大节省存储空间。并且通过 $\boldsymbol{U}$ 可以将其变换回去：

$$\hat{\boldsymbol{f}}=\boldsymbol{U}\boldsymbol{y}+\boldsymbol{u}$$

8.2.2 特征向量的选取

上面的 $\boldsymbol{U}$ 一共有 M 个特征向量。虽然 M 比 N^2 小很多，但通常情况下，M 仍然显得较大。事实上，在实际应用中，并非需要保留所有的 $\boldsymbol{u}_i$。

考虑到使用 K－L 变换作为对人脸图像的压缩手段，可以选取最大的前 k 个特征向量，使得

$$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{M}\lambda_i} \geqslant \alpha$$

在上式中，根据问题需要可以选取适当的 α 值，如选取 $\alpha=99\%$。这说明样本集在前 k 个轴上的信息占全部信息的 99%。

令

$$\hat{\boldsymbol{U}}=(\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_k)$$

则上述对图像的压缩变为

$$\boldsymbol{U}=\hat{\boldsymbol{U}}^{\mathrm{T}}(\boldsymbol{f}-\boldsymbol{u})$$

相应的重建图像为

$$\hat{\boldsymbol{f}}=\hat{\boldsymbol{U}}\boldsymbol{y}+\boldsymbol{u}$$

8.3 2DPCA 方法实现人脸图像压缩与重建

8.3.1 概　述

从 8.2 节对基本的 PCA 实现人脸图像压缩与重建的介绍中得知，二维的人脸图像事先必须转换成一维的向量，而且这一向量的维数往往是很高的。根据一批训练样本实际计算得到的协方差矩阵是对真实协方差矩阵的近似，而训练样本的数量相对协方差矩阵的维数来说往往显得过小，因此，想精确得到协方差矩阵是困难的。虽然实际应用中往往并不直接得到这个协方差矩阵，而是通过其他途径间接得到其特征值和相应的特征向量，但并不说明这样可以精

确地得到特征值和特征向量，因为特征值和特征向量是由协方差矩阵决定的，这和用什么方法得到它们无关。所以，协方差矩阵的不精确必然导致相应的特征值和特征向量的不精确。

为了克服上述缺点，Jian Yang，David Zhang，Alejandro F. Frangi 和 Jing-yu Yang 于 2004 年提出了一种新的用于人脸图像的压缩和重建方法，即 two-dimensional principal component analysis(2DPCA)。这种方法和基本的 PCA 方法不同，它基于二维矩阵而不是一维向量。也就是说，图像矩阵不必事先转换成一个长向量，而是基于原始图像矩阵得到一个图像协方差矩阵。相对于基本的 PCA 方法得到的协方差矩阵，2DPCA 方法得到的图像协方差矩阵的尺寸要小得多。2DPCA 相对于基本的 PCA 主要有两方面的优点：一方面它可以比较精确地得到协方差矩阵；另一方面，2DPCA 求相应的特征向量所用的时间也比基本的 PCA 方法所用的时间短。

下面对其算法思想和具体步骤进行介绍。

8.3.2　2DPCA 算法介绍

假设 $\boldsymbol{X}$ 为一个 n 维归一化的列向量。我们的想法是将图像 $\boldsymbol{A}$（一个 $m\times n$ 的矩阵）按照公式

$$\boldsymbol{Y}=\boldsymbol{A}\boldsymbol{X}$$

投影到 $\boldsymbol{X}$ 上。因而，得到了一个 m 维的投影向量 $\boldsymbol{Y}$，称 $\boldsymbol{Y}$ 为图像 $\boldsymbol{A}$ 的投影特征向量。那么如何得到一个好的投影方向 $\boldsymbol{X}$ 呢？事实上，所有样本投影后的分散程度可以用来衡量投影向量 $\boldsymbol{X}$ 区分样本的能力强弱。投影后的样本越分散，$\boldsymbol{X}$ 就越好。

事实上，可以用投影后的向量的协方差的迹来刻画样本投影后的分散程度，即

$$J(\boldsymbol{X})=\mathrm{tr}(\boldsymbol{S}_x) \tag{8-4}$$

式中，S_x 表示训练样本投影后的向量的协方差；$\mathrm{tr}(\boldsymbol{S}_x)$ 表示 $\boldsymbol{S}_x$ 的迹。

最大化式(8-4)的物理意义在于找到一个投影方向，使得所有训练样本在其上投影后的向量之间的分散程度最大。$\boldsymbol{S}_x$ 的协方差矩阵可以表示为：

$$\begin{aligned}\boldsymbol{S}_x &= \mathrm{E}(\boldsymbol{Y}-\mathrm{E}(\boldsymbol{Y}))(\boldsymbol{Y}-\mathrm{E}(\boldsymbol{Y}))^{\mathrm{T}}=\mathrm{E}[\boldsymbol{AX}-\mathrm{E}(\boldsymbol{AX})][\boldsymbol{AX}-\mathrm{E}(\boldsymbol{AX})]^{\mathrm{T}}\\ &=\mathrm{E}[(\boldsymbol{A}-\mathrm{E}(\boldsymbol{A}))\boldsymbol{X}][(\boldsymbol{A}-\mathrm{E}(\boldsymbol{A}))\boldsymbol{X}]^{\mathrm{T}}\end{aligned}$$

因此

$$\mathrm{tr}(\boldsymbol{S}_x)=\boldsymbol{X}^{\mathrm{T}}[\mathrm{E}(\boldsymbol{A}-\mathrm{E}(\boldsymbol{A}))^{\mathrm{T}}(\boldsymbol{A}-\mathrm{E}\boldsymbol{A})]\boldsymbol{X}$$

可以定义下面的矩阵：

$$\boldsymbol{G}_t=\mathrm{E}[(\boldsymbol{A}-\mathrm{E}(\boldsymbol{A}))^{T}(\boldsymbol{A}-E(\boldsymbol{A}))]$$

矩阵 $\boldsymbol{G}_t$ 被称为图像协方差矩阵。从定义容易知道，$\boldsymbol{G}_t$ 是 $n\times n$ 的非负定矩阵。可以利用训练样本图像直接计算 $\boldsymbol{G}_t$。假设一共有 M 个训练图像样本，$m\times n$ 矩阵 $\boldsymbol{A}_j(j=1,2,\cdots,M)$ 表示第 j 个训练图像，$\overline{\boldsymbol{A}}$ 表示所有训练样本的平均图像。

$$\overline{\boldsymbol{A}}=\frac{1}{M}\sum_{j=1}^{M}\boldsymbol{A}_j$$

那么，$\boldsymbol{G}_t$ 可以写成

$$\boldsymbol{G}_t=\frac{1}{M}\sum_{j=1}^{M}(\boldsymbol{A}_j-\overline{\boldsymbol{A}})^{\mathrm{T}}(\boldsymbol{A}_j-\overline{\boldsymbol{A}}) \tag{8-5}$$

因此，式(8-4)可以改写成

$$J(\boldsymbol{X})=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{G}_t\boldsymbol{X} \tag{8-6}$$

式中，$\boldsymbol{X}$ 为归一的列向量。

式(8-6)为一般总体分散度准则(Generalized Total Scatter Criterion)。使得 $J(X)$ 最大化的归一化向量 $\boldsymbol{X}_{\mathrm{opt}}$ 被称为最优投影轴。也就是说，所有训练图像样本矩阵投影在 $\boldsymbol{X}_{\mathrm{opt}}$ 之后得到的向量之间的分布散度最大。

文献[21]指出，$\boldsymbol{X}_{\mathrm{opt}}$ 是与 $\boldsymbol{G}_t$ 的最大特征值对应的特征向量。一般说来，只有一个最佳投影方向往往是不够的，需要找出一系列的投影方向，即投影方向集 $\{X_1,X_2,\cdots,X_d\}$，满足使得 $J(X)$ 尽量最大化以及彼此正交的原则，即

$$\begin{cases}\{\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d\}=\arg\max J(\boldsymbol{X})\\ \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_j=0,\quad i\neq j,i,j=1,2,\cdots,d\end{cases}$$

事实上，满足上述原则的投影方向集 $\{\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d\}$ 是与 $\boldsymbol{G}_t$ 的前 d 个最大特征值对应的彼此正交的特征向量。

8.3.3 图像压缩(特征提取)

可以利用由2DPCA得到的投影方向集 $\{\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d\}$ 对图像进行压缩。对于一个给定的图像样本 $\boldsymbol{A}$，令

$$\boldsymbol{Y}_k=(\boldsymbol{A}-\overline{\boldsymbol{A}})\boldsymbol{X}_k,\quad k=1,2,\cdots,d$$

这样可以得到一系列的投影向量 $\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots,\boldsymbol{Y}_d$，称之为样本图像 $\boldsymbol{A}$ 的主成分(向量)。这里读者可能已经注意到，用2DPCA方法得到的主成分是一个向量，而基本的PCA方法得到的主成分是标量。

令

$$\boldsymbol{B}=[\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots,\boldsymbol{Y}_d]$$

称其为样本图像 $\boldsymbol{A}$ 的压缩矩阵或者特征矩阵(特征图像)。这就完成了对给定图像的压缩。

8.3.4 图像重建

基本的PCA方法中，可以利用图像压缩后的特征向量和主成分来完成对图像的重建。与之类似，2DPCA可以按照下面的步骤完成压缩后图像的重建。

设属于图像协方差矩阵 $\boldsymbol{G}_t$ 的最大的前 d 个特征值的特征向量为 $\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d$，当图像矩阵在其上投影后，得到的主成分向量为

$$\boldsymbol{Y}_k=(\boldsymbol{A}-\overline{\boldsymbol{A}})\boldsymbol{X}_k,\quad k=1,2,\cdots,d$$

令

$$\boldsymbol{V}=[\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots,\boldsymbol{Y}_d]$$

$$\boldsymbol{U}=[\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d]$$

因此对于给定的样本图像 $\boldsymbol{A}$：

$$\boldsymbol{V}=(\boldsymbol{A}-\overline{\boldsymbol{A}})\boldsymbol{U} \tag{8-7}$$

由于 $\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_d$ 是彼此正交的，所以从式(8-7)容易得到 $\boldsymbol{A}$ 的重建图像：

$$\widetilde{\boldsymbol{A}}=\boldsymbol{V}\boldsymbol{U}^{T}+\overline{\boldsymbol{A}}=\sum_{k=1}^{d}\boldsymbol{Y}_k\boldsymbol{X}_k^{\mathrm{T}}+\overline{\boldsymbol{A}} \tag{8-8}$$

如果令

$$\widetilde{\boldsymbol{A}}_k=\boldsymbol{Y}_k\boldsymbol{X}_k^{\mathrm{T}}+\frac{\overline{\boldsymbol{A}}}{d}$$

我们称 $\tilde{A}_k$ 为 A 的重建子图像，它和 A 以及 $\tilde{A}$ 的尺寸一样。从式(8-8)可以看到，$\tilde{A}$ 可以近似地表示成 k 个子图像之和。重建的精度和之前选择的 G_t 的前 d 个最大特征值的具体 d 值有关，如果 $d=n$（n 为 G_t 的全部特征值个数），那么重建后的 $\tilde{A}=A$；否则，$\tilde{A}$ 是 A 的近似。

8.4 MatPCA 方法实现人脸图像压缩与重建

8.4.1 概　述

MatPCA 方法是由 Songcan Chen，Yulian Zhu，Daoqiang Zhang，Jing-Yu Yang 在 2004 年提出的，将其用在特征提取方面。该方法主要是受 2DPCA 的启发。在处理庞大的向量形态特征时，将其转化成矩阵形态特征来利用 2DPCA 方法对转换后的特征矩阵进行二次特征提取。这样处理在模式识别的时候可以提高识别准确率以及处理速度。

MatPCA 在面对向量形态的特征时会转化成矩阵形态的特征，前提是这个向量的长度必须是合数，否则无法转化成二维矩阵。如果是矩阵形态的特征，那么 MatPCA 就不用提前对其进行处理了。因此对于图像矩阵来说，如果将其看成矩阵特征，那么 MatPCA 方法和 2DPCA 方法是类似的。

对于图像矩阵，MatPCA 方法和 2DPCA 方法是类似的。只不过 2DPCA 是对原始图像矩阵进行行投影，从方差最大化的角度来寻找最优投影方向；而 MatPCA 是对原始图像矩阵进行列投影，从重构误差最小的角度来寻找最优投影方向。两种方法殊途同归，都要对各自的图像协方差矩阵求特征值，并根据实际需要确定选择最大特征值的个数。

8.4.2 MatPCA 算法

设一共有 M 个 $m\times n$ 的训练图像样本 A_j $(j=1,2,\cdots,M)$，训练样本的平均图像为 $\bar{A}$。令

$$X=(x_1,x_2,\cdots,x_d)$$

满足：

$$X^{\mathrm{T}}X=I$$

其中，$x_1,x_2,\cdots,x_d$ 为长为 m 的列向量。对于图像 A_i 我们可以得到如下的线性变换：

$$Y_i=X_i^{\mathrm{T}}(A_i-\bar{A}) \tag{8-9}$$

类似于 2DPCA，在寻找合适的 X 时，需要遵循一定的原则，使得投影后尽可能多的保留原来训练图像样本的信息。一种可行的原则就是采用重建误差原则(Reconstructed Error，RCE)，即最小化：

$$\mathrm{RCE}(X)=\frac{1}{M}\sum_{i=1}^{M}\|A_i-\hat{A}_i\|^2 \tag{8-10}$$

式中，$\|A_i-\hat{A}_i\|$ 表示矩阵 2-范数，$\hat{A}_i=XY_i+\bar{A}$ $(i=1,2,\cdots,M)$。

对于矩阵 A 来说，$\|A\|=\mathrm{tr}(AA^{\mathrm{T}})$，因此

$$\mathrm{RCE}(X)=\frac{1}{M}\sum_{i=1}^{M}\|A_i-\hat{A}_i\|^2=\frac{1}{M}\sum_{i=1}^{M}\|A_i-\hat{A}_i\|^2-\mathrm{tr}(X^{\mathrm{T}}S_t^{\mathrm{Mat}}X) \tag{8-11}$$

其中：

$$S_t^{\mathrm{Mat}}=\frac{1}{M}\sum_{i=1}^{M}(\boldsymbol{A}_i-\overline{\boldsymbol{A}})(\boldsymbol{A}_i-\overline{\boldsymbol{A}})^{\mathrm{T}}$$

S_t^{Mat} 称为由给定样本矩阵得到的总体协方差矩阵。

从式(8-11)可以明显看出，只需要最大化式(8-12)即可：

$$J(\boldsymbol{X})=\mathrm{tr}(\boldsymbol{X}^{\mathrm{T}}S_t^{\mathrm{Mat}}\boldsymbol{X}) \tag{8-12}$$

式(8-12)等价于求解如下矩阵方程：

$$S_t^{\mathrm{Mat}}\boldsymbol{X}=\boldsymbol{X}\boldsymbol{\Lambda} \tag{8-13}$$

其中，$\boldsymbol{\Lambda}=\mathrm{diag}(\lambda_1,\lambda_2,\cdots,\lambda_m)$是对角矩阵，其对角元素即 S_t^{Mat} 的所有非负特征值，$\boldsymbol{X}$是由特征向量组成的矩阵。

接下来在选取特征值的时候可以参考基本的 PCA 方法以及 2DPCA 方法，即选取适当的 α 值，使得：

$$\frac{\sum_{i=1}^{d}\lambda_i}{\sum_{i=1}^{M}\lambda_i}\geqslant\alpha$$

在确定合适的 d 之后进而得到由相应的特征向量组成的投影矩阵 $\boldsymbol{X}$。

8.5 ModulePCA 方法实现人脸图像压缩与重建

8.5.1 概 述

ModulePCA 方法是由 Rajkiran Gottumukkal, Vijayan K. Asari 在 2003 年提出的。在利用 ModulePCA 方法对图像进行压缩与重建时主要是将每个训练图像都划分成一些尺寸大小都一样的子图像，然后将所有训练图像的所有子图像集合在一起进行 PCA 分析，得到相应的总体协方差矩阵。

在对测试图像进行压缩时，事先按照训练图像那样的划分方法将测试图像划分成子图像，然后逐个对子图像进行压缩。重建时也是逐个对压缩的子图像进行重建，然后再拼接成原来的图像。

8.5.2 ModulePCA 算法

设一共有 M 个 $m\times n$ 的训练图像样本 $\boldsymbol{A}_j\ (j=1,2,\cdots,M)$，令 m_1,n_1 分别为图像纵向和横向均匀划分的个数。那么每个测试图像均可划分成 $m_1\times n_1$ 个子图像 $\boldsymbol{A}_{jkl}\ (j=1,2,\cdots,M,k=1,2,\cdots,m_1,l=1,2,\cdots,n_1)$。

令

$$\overline{\boldsymbol{A}}_s=\frac{1}{Mm_1n_1}\sum_{j=1}^{M}\sum_{k=1}^{m_1}\sum_{l=1}^{n_1}\boldsymbol{A}_{jkl}$$

为所有训练图像所有子图像的均值，计算所有子图像的总体协方差矩阵：

$$\boldsymbol{S}_s=\frac{1}{Mm_1n_1}\sum_{j=1}^{M}\sum_{k=1}^{m_1}\sum_{l=1}^{n_1}(\boldsymbol{A}_{jkl}-\overline{\boldsymbol{A}}_s)(\boldsymbol{A}_{jkl}-\overline{\boldsymbol{A}}_s)^{\mathrm{T}}$$

选取适当的 α 值，求 S_s 的特征值使得：

$$\frac{\sum_{i=1}^{d}\lambda_i}{\sum_{i=1}^{M}\lambda_i} \geqslant \alpha$$

在确定合适的 d 之后进而得到由相应的特征向量组成的投影矩阵：$X=[x_1,x_2,\cdots,x_d]$。对于测试图像 T，将其按照训练图像划分模式划分成：

$$T_{kl}\ (k=1,2,\cdots,m_1;l=1,2,\cdots,n_1)$$

计算

$$Y_{kl}=X^{\mathrm{T}}(T_{kl}-\overline{A}_s)$$

完成对测试图像各个子图像的压缩。

计算

$$\widetilde{T}_{kl}=XY_{kl}+\overline{A}_s$$

完成对测试图像各个子图像的重建，将重建后的子图像拼接在一起完成对 T 的重建。

8.6 算法在 MATLAB 平台上的实现

8.6.1 概　述

本章前半部分对各种 PCA 方法对人脸图像压缩与重建进行了理论方面的介绍。这节着重给出算法在 MATLAB 平台上实现的代码。为了方便读者操作，算法采用 GUI(Graphical User Interface)的形式给出，并给出关键代码部分的注释，方便阅读。GUI 采取了代码编写的方式而没有采取 GUIDE(GUI Design Enviornment)方式，一方面是由于笔者更习惯于用代码编写 GUI 这种形式，另一方面还考虑到程序在不同 MATLAB 版本上运行的方便性。

本节一共有两个 GUI 程序，一个是用基本的 PCA 方法与 2DPCA 和 MatPCA 方法实现图像压缩和重建的对比；另一个是用 ModulePCA 方法实现图像压缩和重建。

8.6.2 基本 PCA 与 2DPCA 和 MatPCA 方法 GUI

源代码如下：

```
function PcaMethodsforImageRebuilt
% 主界面，返回主界面句柄
fig = figure('defaultuicontrolunits','normalized','name',...
    '各类 Pca 方法用于图像压缩与重建','numbertitle','off','menubar','none');
AxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on'); % 建立坐标轴，并返回句柄
UiButtonGroupH = uibuttongroup('Position',[0.81 0.08 0.185 0.82],'title',...
    '各 Pca 方法','fontsize',10); % 群组对象，并返回句柄
PcaH = uicontrol('Style','Radio','String','Pca','pos',[0.05 0.8 0.9 0.15],...
    'parent',UiButtonGroupH); % Pca 的选项
TwoDPcaH = uicontrol('Style','Radio','String','2D Pca','pos',...
    [0.05 0.6 0.9 0.15],'parent',UiButtonGroupH); % TwoDPca 的选项
```

```
MatPcaH = uicontrol('Style','Radio','String','Mat Pca','pos',...
     [0.05 0.4 0.9 0.15],'parent',UiButtonGroupH); % MatPca的选项
% 下面几行建立供应的菜单项以及噪声按钮。各菜单以及按钮单击时的回调函数
TrainImagesH = uimenu(fig,'label','训练图像','tag','训练图像',...
     'callback',@TrainImage);
TestImageH = uimenu(fig,'label','测试图像','tag','测试图像',...
     'callback',@TestImage,'enable','off');
SaltPepperNoiseH = uicontrol(fig,'style','pushbutton','Pos',...
     [0.15,0.88,0.2,0.08],'string','添加椒盐噪声','fontsize',10,...
     'enable','off','callback',@SaltPepperNoise);
GaussianNoiseH = uicontrol(fig,'style','pushbutton','Pos',[0.55,0.88,0.2,0.08],...
     'string','添加高斯噪声','fontsize',10,'enable','off','callback',@GaussianNoise);
CompressImageH = uimenu(fig,'label','压缩图像','tag','压缩图像','callback',...
     @CompressImage,'enable','off');
RebuiltImageH = uimenu(fig,'label','重建图像','tag','重建图像','callback',...
     @RebuiltImage,'enable','off');
ImaOutH = uimenu(fig,'label','导出图像到新 figure','callback',@ImaOut);

%====================================================================
% function TrainImage,训练图像函数,根据所选中的 pca 方法训练图像
%====================================================================
    function TrainImage(hObject,eventdata)
        [filename, pathname, flag] = uigetfile('*.bmp','训练图像样本(*.bmp)',...
           'MultiSelect','on','请导入训练图像'); % 弹出提示对话框
        if ~flag
            return; % flag为0,用户取消选择,返回;
        end
        % 将导入的图像样本名称用 cell 数组存储,因为导入一个图像时 filename 用 char 型存储,
        % cellstr 作用后变为 filename 用 1*1 的 cell 数组存储。导入多个图像时 filename 用 cell
        % 存储,cellstr 作用后不变
        filename = cellstr(filename);
        n = length(filename);
        A = cell(n,1); % 存储导入的训练图像
        if n == 1,
            A{1} = imread([pathname filename]); % 个数为 1 时候的训练图像读取
        else
            for k = 1:n
                A{k} = imread([pathname filename{k}]); % 个数为 n 时候的训练图像读取
            end
        end
        percent = 0.99; % 各个主成分分析中希望保留的信息百分比
        % 当前图像中选择的是哪种 Pca
        WhichPca = get(findobj(UiButtonGroupH,'style','radio','value',1),'string');
        switch WhichPca
            case 'Pca' % 该 case 是选择基本 Pca 方法时候的训练过程。下面 case 类推
                A1 = zeros(numel(A{1}),n);
                for k = 1:n
                    A1(:,k) = A{k}(:); % 存储拉伸成向量后的各个训练图像
                end
                Abar = mean(A1,2); % 各个训练图像的均值
```

```
        X = A1 - repmat(Abar,1,n); % 每个训练图像都减去均值
        R = X' * X; % 得到要求解特征值的矩阵
        % 求解特征值特征向量。EigV 是特征向量,Lamda 是特征值组成的对角矩阵。
        [EigV,Lamda] = eig(R);
        diagL = diag(Lamda); % 提取出特征向量
        % 特征值由大到小排序,ix 是排好序的值在原来序列中的序号
        [diagL1,ix] = sort(diagL,'descend');
        EigV = EigV(:,ix); % 特征向量相应排序
        cumsumL1 = cumsum(diagL1); % 累加特征值
        % 找到累加比重大于 percent 的第一个元素位置
        ind = find(cumsumL1/cumsumL1(end)> = percent,1,'first');
        Lamda = diagL1(1:ind); % 提取出相应的特征值
        EigV = EigV(:,1:ind); % 提取出相应的特征向量
        % 根据书中相关公式变换回需要的特征向量
        EigV = X * EigV./repmat(sqrt(Lamda'),size(X,1),1);
        Pca.Lamda = Lamda;
        Pca.EigV = EigV;
        Pca.Abar = Abar;
        % 存储压缩时用到的相关数据到句柄值为 TrainImagesH 的对象中去。
        setappdata(TrainImagesH,'Pca',Pca);
    case '2D Pca'
        Abar = zeros(size(A{1})); % 将来用来存储导入的训练图像的平均值

        % 将来用来存储导入的训练图像的协方差矩阵,即 Two - Dimensional PCA:
        % A New Approach to Appearance - Based Face Representation and
        % Recognition 文献中的 Gt
        S = zeros(size(A{1},2));
        for k = 1:n
            Abar = Abar + double(A{k});
        end
        Abar = Abar/n; % 求出来的平均值
        for k = 1:n
            S = S + (double(A{k})- Abar)' * (double(A{k})- Abar);
        end
        S = S/n; % 求出来的协方差矩阵
        % 求解特征值特征向量。EigV 是特征向量,Lamda 是特征值组成的对角矩阵
        [EigV,Lamda] = eig(S);
        diagL1 = diag(Lamda); % 提取出特征向量
        % 特征值由大到小排序,ix 是排好序的值在原来序列中的序号
        [diagL1,ix] = sort(diagL1,'descend');
        EigV = EigV(:,ix); % 特征向量相应排序
        cumsumL1 = cumsum(diagL1); % 累加特征值
        % 找到累加比重大于 percent 的第一个元素位置
        ind = find(cumsumL1/cumsumL1(end)> = percent,1,'first');
        EigV = EigV(:,1:ind); % 提取出相应的特征值
        Lamda = diagL1(1:ind); % 提取出相应的特征向量
        TwoDPca.Lamda = Lamda;
        TwoDPca.EigV = EigV;
        TwoDPca.Abar = Abar;
        % 存储压缩时用到的相关数据到句柄值为 TrainImagesH 的对象中去。
```

```
            setappdata(TrainImagesH,'TwoDPca',TwoDPca);
        case 'Mat Pca'
            Abar = zeros(size(A{1})); % 将来用来存储导入的训练图像的平均值
            S = zeros(size(A{1},1)); % 将来用来存储导入的训练图像的协方差矩阵,
            for k = 1:n
                Abar = Abar + double(A{k});
            end
            Abar = Abar/n; % 求出来的平均值
            for k = 1:n
                S = S + (double(A{k})- Abar) * (double(A{k})- Abar)';
            end
            S = S/n; % 求出来的协方差
            [EigV,Lamda] = eig(S); % 以下功能类似'2D Pca'中的。
            diagLl = diag(Lamda);
            [diagLl,ix] = sort(diagLl,'descend');
            EigV = EigV(:,ix);
            cumsumLl = cumsum(diagLl);
            ind = find(cumsumLl/cumsumLl(end)>= percent,1,'first');
            EigV = EigV(:,1:ind);
            Lamda = diagLl(1:ind);
            MatPca.Lamda = Lamda;
            MatPca.EigV = EigV;
            MatPca.Abar = Abar;
            setappdata(TrainImagesH,'MatPca',MatPca);
        otherwise
    end
    h = msgbox('已训练好,请选择测试图像'); % 弹出对话框
    uiwait(h); % 等待用户确认后再运行程序
    set(TestImageH,'enable','on'); % 设置测试图像菜单状态为能用状态
end

%=========================================================
% function TestImage,选择一个测试图像并显示出来
%=========================================================
function TestImage(hObject,eventdata)
    [filename, pathname, flag] = uigetfile('*.bmp','测试图像(*.bmp)',...
        '请选择一个测试图像'); % 弹出选择测试图像对话图框
    if ~flag
        return; % flag为0,用户取消选择,返回;
    end
    imshow([pathname,filename]); % 显示选择的图像
    set([CompressImageH,SaltPepperNoiseH,GaussianNoiseH],...
        'enable','on'); % 设置压缩图像菜单状态、椒盐噪声按钮、高斯噪声按钮为能用状态
    ImageData.pathname = pathname;
    ImageData.filename = filename;
    ImageData.data = imread([pathname,filename]);
    title(ImageData.filename);
    % 存储相关数据到测试图像菜单这个对象中去。
    setappdata(TestImageH,'ImageData',ImageData);
end
```

```
%=====================================================
% function SaltPepperNoise,添加椒盐噪声
%=====================================================
    function SaltPepperNoise(hObject,eventdata)
        Prompt = '请输入噪声密度 d:';
        Name = '椒盐噪声参数输入';
        Numlines = 1; % 行数
        DefaultAnswer = {'0.05'}; % 默认值
        Answer = inputdlg(Prompt,Name,Numlines,DefaultAnswer); % 输入对话框
        % 与 setappdata 对应,获取先前存储的数据。
        ImageData = getappdata(TestImageH,'ImageData');
        ImageData.data = imnoise(imread([ImageData.pathname,ImageData.filename]),...
            'salt & pepper',str2double(Answer{1})); % 添加椒盐噪声
        imshow(ImageData.data); % 显示添加椒盐噪声后的图像
        ImageData.name = [ImageData.filename,'添加椒盐噪声 噪声密度 ',Answer{1}];
        title(ImageData.name);
        % 存储添加椒盐噪声后的图像相关数据
        setappdata(TestImageH,'ImageData',ImageData);
    end
%=====================================================
% function GaussianNoise,添加高斯噪声(注释完全类似添加椒盐噪声)
%=====================================================
    function GaussianNoise(hObject,eventdata)
        Prompt = {'请输入噪声均值:','请输入噪声方差:'};
        Name = '高斯噪声参数输入';
        Numlines = 1;
        DefaultAnswer = {'0','0.01'};
        Answer = inputdlg(Prompt,Name,Numlines,DefaultAnswer);
        ImageData = getappdata(TestImageH,'ImageData');
        ImageData.data = imnoise(imread([ImageData.pathname,ImageData.filename]),...
            'gaussian',str2double(Answer{1}),str2double(Answer{2}));
        imshow(ImageData.data);
        ImageData.name = [ImageData.filename,'添加高斯噪声 均值 ',...
            Answer{1},'方差 ',Answer{2}];
        title(ImageData.name);
        setappdata(TestImageH,'ImageData',ImageData);
    end
%==========================================================
% function CompressImage,压缩图像
%==========================================================
    function CompressImage(hObject,eventdata)
        ImageData = getappdata(TestImageH,'ImageData'); % 获取先前存储的测试图像信息
        Ima = ImageData.data; % Imagedata 是结构数组类型,data 域存储的是图像数据
        WhichPca = get(findobj(UiButtonGroupH,'style','radio','value',1),...
            'string'); % 判断当前选择的是哪种主成分分析方法
        switch WhichPca
            case 'Pca'
                try
                    % 获取训练图像时存储的训练结果相关数据
                    Pca = getappdata(TrainImagesH,'Pca');
```

```
            %乘以特征向量组成的主成分矩阵压缩图像!
            Y = Pca.EigV' * (double(Ima(:))- Pca.Abar);
        catch lasterr
            %如果有错误(一般就是没有训练图像而用户单击了这个菜单项,
            %运行下面的提示信息)
            h = msgbox('尚未用 Pca 算法训练图像,请训练图像');
            uiwait(h);
            return;
        end
        %以下提示压缩完毕并保存压缩结果
        hstr = '当前图像已用 Pca 算法压缩完毕,请按提示保存结果以供重建调用';
        h = helpdlg(hstr,'Pca 算法图像压缩');
        uiwait(h);
        [file,path] = uiputfile(['Pca',get(get(gca,'title'),'string'),...
            '.mat'],'保存当前压缩图像为:');
        if~file
            return;
        end
        save([path,file],'Y');
        %'2D Pca'和'Mat Pca'的注释参照'Pca'的
    case '2D Pca'
        try
            TwoDPca = getappdata(TrainImagesH,'TwoDPca');
            Y = (double(Ima)- TwoDPca.Abar) * TwoDPca.EigV;
        catch lasterr
            h = msgbox('尚未用 2DPca 算法训练图像,请训练图像');
            uiwait(h);
            return;
        end
        hstr = '当前图像已用 2DPca 算法压缩完毕,请按提示保存结果以供重建调用';
        h = helpdlg(hstr,'2DPca 算法图像压缩');
        uiwait(h);
        [file,path] = uiputfile(['TwoDPca',get(get(gca,'title'),...
            'string'),'.mat'],'保存当前压缩图像为:');
        if ~file
            return;
        end
        save([path,file],'Y');
    case 'Mat Pca'
        try
            MatPca = getappdata(TrainImagesH,'MatPca');
            Y = MatPca.EigV' * (double(Ima)- MatPca.Abar);
        catch lasterr
            h = msgbox('尚未用 MatPca 算法训练图像,请训练图像');
            uiwait(h);
            return;
        end
        hstr = '当前图像已用 MatPca 算法压缩完毕,请按提示保存结果以供重建调用';
        h = helpdlg(hstr,'MatPca 算法图像压缩');
        uiwait(h);
```

```
                [file,path] = uiputfile(['MatPca',get(get(gca,'title'),...
                    'string'),'.mat'],'保存当前压缩图像为:');
                if ~file
                    return;
                end
                save([path,file],'Y');
            otherwise
        end
        set(RebuiltImageH,'enable','on');
    end
%=================================================================
% function RebuiltImage,图像重建
%=================================================================
    function RebuiltImage(hObject,eventdata)
        % 获取存储的测试原图像数据,主要是计算重建图像与原图像误差时用到
        ImageData = getappdata(TestImageH,'ImageData');
        Ima = ImageData.data;
        WhichPca = get(findobj(UiButtonGroupH,'style','radio','value',1),'string');
        switch WhichPca
            case 'Pca'
                [filename, pathname, flag] = uigetfile(  '*.mat',...
                '请导入 Pca 算法压缩的图像矩阵');
            if ~flag
                return; % flag 为 0,用户取消选择,返回;
            end
            a = load([pathname filename]);
            Y = a.Y;
            % 获取存储的训练图像数据(里面有特征值、特征向量等必要数据)
            Pca = getappdata(TrainImagesH,'Pca');
            Rebuilt = double(Pca.EigV * Y) + Pca.Abar; % 图像重建
            figure('defaultuicontrolunits','normalized');
            NewAxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on');
            % 显示重建图像
            imshow(reshape(uint8(Rebuilt),size(Ima,1),size(Ima,2)));
            title([get(get(AxesH,'title'),'string'),':用 Pca 算法重建后图像'])
            % 计算重建图像与原图像误差
            Mse = sqrt(mean((Rebuilt - double(Ima(:))).^2 ));
            % 界面上显示结果
            uicontrol(gcf,'style','text','pos',[0.71  0.5  0.28,0.1],'string',...
                ['重建图像与原图像均方根误差:',num2str(Mse)],'fontsize',10);

            % '2D Pca'和'Mat Pca'的重建注释参照'Pca'的
        case '2D Pca'
            [filename, pathname, flag] = uigetfile('*.mat',...
                '请导入 TwoDPca 算法压缩的图像矩阵');
            if ~flag
                return; % flag 为 0,用户取消选择,返回;
            end
            a = load([pathname filename]);
            Y = a.Y;
```

```
                TwoDPca = getappdata(TrainImagesH,'TwoDPca');
                Rebuilt = Y * TwoDPca.EigV' + TwoDPca.Abar;
                figure('defaultuicontrolunits','normalized');
                NewAxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on');
                imshow(uint8(Rebuilt));
                title([get(get(AxesH,'title'),'string'),':用 2DPca 算法重建后图像'])
                Mse = sqrt(mean(mean((Rebuilt - double(Ima)).^2 )) );
                uicontrol(gcf,'style','text','pos',[0.71 0.5 0.28,0.1],'string',...
                    ['重建图像与原图像均方根误差:',num2str(Mse)],'fontsize',10);
            case 'Mat Pca'
                [filename, pathname, flag] = uigetfile(  '*.mat',...
                    '请导入 MatPca 算法压缩的图像矩阵');
                if ~flag
                    return; % flag 为 0,用户取消选择,返回;
                end
                a = load([pathname filename]);
                Y = a.Y;
                MatPca = getappdata(TrainImagesH,'MatPca');
                Rebuilt = MatPca.EigV * Y + MatPca.Abar;
                figure('defaultuicontrolunits','normalized');
                NewAxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on');
                imshow(uint8(Rebuilt));
                title([get(get(AxesH,'title'),'string'),':用 MatPca 算法重建后图像'])
                Mse = sqrt(mean(mean((Rebuilt - double(Ima)).^2 )) );
                uicontrol(gcf,'style','text','pos',[0.71 0.5 0.28,0.1],'string',...
                    ['重建图像与原图像均方根误差:',num2str(Mse)],'fontsize',10);
                otherwise
            end
        end
%====================================================================
% function ImaOut,导出主界面坐标轴中的图像,便于保存
%====================================================================
        function ImaOut(hObject,eventdata)
            NewFig = figure;
            NewAxesH = copyobj(AxesH,NewFig); % 复制旧坐标轴对象到新的 figure
            set(NewAxesH,'position',[0.1300 0.1100 0.7750 0.8150]);
            set(NewFig,'colormap',get(fig,'colormap'));
        end
end
```

运行上述程序得到图 8.1 所示界面。右边有三个单选按钮，分别为 PCA 方法、2DPCA 方法和 MatPCA 方法对图像进行压缩和重建，默认为 PCA 方法。然而，界面一些菜单选项和按钮处于不可用状态，这是由算法处理流程决定的。首先要选择“训练图像”菜单项来选取训练图像样本。这里我们选择的是 ORL 数据库。ORL 数据库可以在随书免费附赠的源代码中找到。

选择好“训练图像”后，会弹出提示对话框，提示选择测试图像。这时候“测试图像”菜单项变为可用状态，接下来选择接受压缩的测试图像，譬如选择名为 ORL001.bmp 的图像，得到图 8.2 所示界面。这时候，“添加椒盐噪声”和“高斯噪声”按钮以及“压缩图像”菜单项就可用了，可以直接压缩图像，也可以添加噪声后再压缩图像，如图 8.3 所示。压缩图像后会提示保存压

缩后的图像，保存完毕后，"重建图像"菜单项可以使用，以供重建图像时使用如图 8.4 所示。

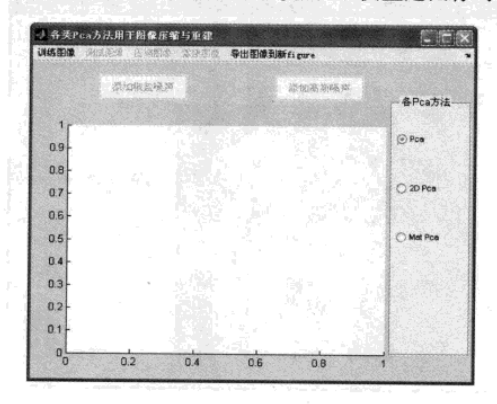


图 8.1　运行 PcaMethodsforImageRebuilt 得到的界面



图 8.2　选择好测试图像后得到的界面



图 8.3　添加高斯噪声后的测试图像



图 8.4　用 PCA 方法重建后的的测试图像

读者可以自行操作该 GUI 进行实验，GUI 的回调函数都以 nested function 的形式给出，方便数据共享。此外还可以多次利用 setappdata 和 getappdata 进行 GUI 不同控件之间的数据传递。

8.6.3　Module PCA 方法 GUI

源代码如下：

```
function ModuleforImageRebuilt
fig = figure('defaultuicontrolunits','normalized','name',...
    'ModulePca 方法用于图像压缩与重建','numbertitle','off','menubar','none');
AxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on');
TrainImagesH = uimenu(fig,'label','训练图像','tag','训练图像',...
      'callback',@TrainImage);
TestImageH = uimenu(fig,'label','测试图像','tag','测试图像','callback',...
      @TestImage,'enable','off');
SaltPepperNoiseH = uicontrol(fig,'style','pushbutton','Pos',...
      [0.1,0.88,0.2,0.08],'string','添加椒盐噪声','fontsize',10,...
      'enable','off','callback',@SaltPepperNoise);
GaussianNoiseH = uicontrol(fig,'style','pushbutton','Pos',...
      [0.55,0.88,0.2,0.08],'string','添加高斯噪声','fontsize',10,...
      'enable','off','callback',@GaussianNoise);
CompressImageH = uimenu(fig,'label','压缩图像','tag','压缩图像',...
      'callback',@CompressImage,'enable','off');
RebuiltImageH = uimenu(fig,'label','重建图像','tag','重建图像',...
      'callback',@RebuiltImage,'enable','off');
ImaOutH = uimenu(fig,'label','导出图像到新 figure','callback',@ImaOut);
%======================================================================
% function TrainImage,ModulePCA 方法训练图像函数
%======================================================================
      function TrainImage(hObject,eventdata)
```

```
[filename, pathname, flag] = uigetfile('*.bmp','训练图像样本(*.bmp)',...
    'MultiSelect','on','请导入训练图像');
if ~flag
    return; % flag 为 0,用户取消选择,返回;
end
% 将导入的图像样本名称用 cell 数组存储,因为导入一个图像时 filename 用 char 型存储,
% cellstr 作用后变为 filename 用 1*1 的 cell 数组存储。导入多个图像时 filename 用 cell
% 存储,cellstr 作用后不变
filename = cellstr(filename);
n = length(filename);
A = cell(n,1); % 存储导入的训练图像
if n == 1,
    A{1} = imread([pathname filename]);
else
    for k = 1:n,
        A{k} = imread([pathname filename{k}]);
    end
end
percent = 0.98; % 希望保留的信息百分比
[Nrow,Ncol] = size(A{1}); % 返回图像的尺寸
Prompt = {['图像纵向划分个数(能整除',num2str(Nrow),')'],...
    ['图像横向划分个数(能整除',num2str(Ncol),')']};
Name = '模块划分数输入';
Numlines = 1;
DefaultAnswer = {'2','2'};
Answer = inputdlg(Prompt,Name,Numlines,DefaultAnswer); % 输入对话框
if isempty(Answer)
    return; % 如果用户取消就返回
end
RowDivideNum = str2double(Answer{1}); % RowDivideNum:行方向上分的块数。
ColDivideNum = str2double(Answer{2}); % ColDivideNum:列方向上分的块数
if mod(Nrow,RowDivideNum)||mod(Ncol,ColDivideNum)
    h = msgbox(['输入的划分个数不能整除图像的高:',num2str(Nrow),...
        '或者宽:',num2str(Ncol)]); % 不能整除就报错。
    uiwait(h);
    return;
end
% 三维矩阵,前两维是每个训练图像划分成模块后,这些模块的平均图像。
% 第三维表示训练图像序号
MeanModuleImage = zeros(Nrow/RowDivideNum,Ncol/ColDivideNum,n);
% 四维 cell 数组,存储划分成模块后的所有训练图像。最后一维表示训练图像序号。
% 前两维之所以为长 1,是为了计算方便处理。
ModuleImage = cell(1,1,RowDivideNum * ColDivideNum,n);
% 存储每个训练图像的平均协方差矩阵
MeanModuleCov = zeros(Nrow/RowDivideNum,Nrow/RowDivideNum,n);
for k = 1:n
    Temp = mat2cell(double(A{k}),Nrow/RowDivideNum * ones(1,RowDivideNum),...
        Ncol/ColDivideNum * ones(1,ColDivideNum)); % 第 k 个训练图像模块划分

    % 将划分成模块的图像组成三维矩阵(即各个模块叠加在一起)
```

```
            Module = cell2mat(reshape(Temp,1,1,RowDivideNum * ColDivideNum));
            MeanModuleImage(:,:,k) = mean(Module,3); % 求第 k 个训练图像模块的平均值
            % 将第 k 个模块图像改成 1 * 1 * (RowDivideNum * ColDivideNum)的数组保存到
            % ModuleImage 里。
            ModuleImage(:,:,:,:,k) = reshape(Temp,1,1,RowDivideNum * ColDivideNum);
        end
        Abar = mean(MeanModuleImage,3); % 求所有训练图像的模块图像的平均值。
        for k = 1:n
            % 计算每个训练图像的平均协方差矩阵
            MeanModuleCov(:,:,k) = mean( cell2mat( cellfun(@(x) (x - Abar) * ...
                (x - Abar)',ModuleImage(1,1,:,k),'UniformOutput',false)) ,3);
        end
        S = mean(MeanModuleCov,3); % 计算所有训练图像的平均协方差矩阵的平均值
        % 以下步骤类似前面几种 Pca
        [EigV,Lamda] = eig(S);
        diagLl = diag(Lamda);
        [diagLl,ix] = sort(diagLl,'descend');
        EigV = EigV(:,ix);
        cumsumLl = cumsum(diagLl);
        ind = find(cumsumLl/cumsumLl(end)> = percent,1,'first');
        EigV = EigV(:,1:ind);
        Lamda = diagLl(1:ind);
        ModulePca.Lamda = Lamda;
        ModulePca.EigV = EigV;
        ModulePca.Abar = Abar;
        setappdata(TrainImagesH,'ModulePca',ModulePca);
        setappdata(TrainImagesH,'RowDivideNum',RowDivideNum);
        setappdata(TrainImagesH,'ColDivideNum',ColDivideNum);
        h = msgbox('已训练好,请选择测试图像');
        uiwait(h);
        set(TestImageH,'enable','on');
    end

%========================================================
% function TestImage,选择一个测试图像并显示出来
%========================================================
    function TestImage(hObject,eventdata)
        [filename, pathname, flag] = uigetfile('*.bmp','测试图像(*.bmp)',...
            '请选择一个测试图像');
        if ~flag
            return; % flag 为 0,用户取消选择,返回;
        end
        imshow([pathname,filename]); % 显示测试图像
        % 相关按钮状态能用
        set([CompressImageH,SaltPepperNoiseH,GaussianNoiseH],'enable','on');
        % 下面一些主要是为了在图像上按训练时剖分的模块划虚线,方便展示。
        ImageData.pathname = pathname;
        ImageData.filename = filename;
        ImageData.data = imread([pathname,filename]);
        RowDivideNum = getappdata(TrainImagesH,'RowDivideNum');
```

```
        ColDivideNum = getappdata(TrainImagesH,'ColDivideNum');
        [Nrow,Ncol] = size(ImageData.data);
        xrow = repmat([0,Ncol],RowDivideNum - 1,1);
        yrow = linspace(0,Nrow,RowDivideNum + 1);
        yrow([1,end]) = [];
        yrow = [yrow' yrow'];
        xcol = linspace(0,Ncol,ColDivideNum + 1);
        ycol = repmat([0,Nrow],ColDivideNum - 1,1);
        xcol([1,end]) = [];
        xcol = [xcol' xcol'];
        hold on
        for k = 1:ColDivideNum - 1
            plot(xcol(k,:),ycol(k,:),'k--');
        end
        for k = 1:RowDivideNum - 1
            plot(xrow(k,:),yrow(k,:),'k--');
        end
        title(ImageData.filename);
        hold off
        setappdata(TestImageH,'ImageData',ImageData);
    end
%=====================================================
% function SaltPepperNoise,添加椒盐噪声
%=====================================================
    function SaltPepperNoise(hObject,eventdata)
        Prompt = '请输入噪声密度 d:';
        Name = '椒盐噪声参数输入';
        Numlines = 1;
        DefaultAnswer = {'0.05'};
        Answer = inputdlg(Prompt,Name,Numlines,DefaultAnswer);
        if isempty(Answer)
            return;
        end
        ImageData = getappdata(TestImageH,'ImageData');
        ImageData.data = imnoise(imread([ImageData.pathname,ImageData.filename]),...
            'salt & pepper',str2double(Answer{1}));
        imshow(ImageData.data);
        % 下面一些主要是为了在添加椒盐噪声后的图像上按训练时剖分的模块划虚线，方便展示。
        RowDivideNum = getappdata(TrainImagesH,'RowDivideNum');
        ColDivideNum = getappdata(TrainImagesH,'ColDivideNum');
        [Nrow,Ncol] = size(ImageData.data);
        xrow = repmat([0,Ncol],RowDivideNum - 1,1);
        yrow = linspace(0,Nrow,RowDivideNum + 1);
        yrow([1,end]) = [];
        yrow = [yrow' yrow'];
        xcol = linspace(0,Ncol,ColDivideNum + 1);
        ycol = repmat([0,Nrow],ColDivideNum - 1,1);
        xcol([1,end]) = [];
        xcol = [xcol' xcol'];
        hold on
```

```
        for k = 1:ColDivideNum - 1
            plot(xcol(k,:),ycol(k,:),'k--');
        end
        for k = 1:RowDivideNum - 1
            plot(xrow(k,:),yrow(k,:),'k--');
        end
        hold off

        ImageData.name = [ImageData.filename,'添加椒盐噪声 噪声密度 ',Answer{1}];
        title(ImageData.name);
        setappdata(TestImageH,'ImageData',ImageData);
    end

%==========================================================
% function GaussianNoise,添加高斯噪声
%==========================================================
    function GaussianNoise(hObject,eventdata)
        Prompt = {'请输入噪声均值:','请输入噪声方差:'};
        Name = '高斯噪声参数输入';
        Numlines = 1;
        DefaultAnswer = {'0','0.01'};
        Answer = inputdlg(Prompt,Name,Numlines,DefaultAnswer);
        if isempty(Answer)
            return;
        end
        ImageData = getappdata(TestImageH,'ImageData');
        ImageData.data = imnoise(imread([ImageData.pathname,ImageData.filename]),...
            'gaussian',str2double(Answer{1}),str2double(Answer{2}));
        imshow(ImageData.data);
        %%下面一些主要是为了在添加高斯噪声后的图像上按训练时剖分的模块划虚线,方便展示。
        RowDivideNum = getappdata(TrainImagesH,'RowDivideNum');
        ColDivideNum = getappdata(TrainImagesH,'ColDivideNum');
        [Nrow,Ncol] = size(ImageData.data);
        xrow = repmat([0,Ncol],RowDivideNum - 1,1);
        yrow = linspace(0,Nrow,RowDivideNum + 1);
        yrow([1,end]) = [];
        yrow = [yrow' yrow'];
        xcol = linspace(0,Ncol,ColDivideNum + 1);
        ycol = repmat([0,Nrow],ColDivideNum - 1,1);
        xcol([1,end]) = [];
        xcol = [xcol' xcol'];
        hold on
        for k = 1:ColDivideNum - 1
            plot(xcol(k,:),ycol(k,:),'k--');
        end
        for k = 1:RowDivideNum - 1
            plot(xrow(k,:),yrow(k,:),'k--');
        end
        hold off
```

```
        ImageData.name = [ImageData.filename,'添加高斯噪声 均值',Answer{1},...
            '方差',Answer{2}];
        title(ImageData.name);
        setappdata(TestImageH,'ImageData',ImageData);
    end

%==============================================================
% function CompressImage,压缩图像
%==============================================================
    function CompressImage(hObject,eventdata)
        ImageData = getappdata(TestImageH,'ImageData');
        Ima = ImageData.data;
        ModulePca = getappdata(TrainImagesH,'ModulePca');
        RowDivideNum = getappdata(TrainImagesH,'RowDivideNum');
        ColDivideNum = getappdata(TrainImagesH,'ColDivideNum');
        % 以上获取先前存储的相关数据
        [Nrow,Ncol] = size(Ima);
        ImaDivide = mat2cell(double(Ima),Nrow/RowDivideNum * ones(1,RowDivideNum),...
            Ncol/ColDivideNum * ones(1,ColDivideNum)); % 图像剖分
        CompressIma = cellfun(@(x) ModulePca.EigV' * (x - ModulePca.Abar),ImaDivide,...
            'UniformOutput',false); % 各个模块压缩
        h = helpdlg('当前图像已用 Module 算法压缩完毕,请按提示保存结果以供重建调用',...
            'Module 算法图像压缩');
        uiwait(h);
        [file,path] = uiputfile(['Module',...
            get(get(gca,'title'),'string'),'.mat'],'保存当前压缩图像为:');
        if~file
            return;
        end
        save([path,file],'CompressIma');
        set(RebuiltImageH,'enable','on');
    end
%==================================================================
% function RebuiltImage,图像重建
%==================================================================
    function RebuiltImage(hObject,eventdata)
        ImageData = getappdata(TestImageH,'ImageData');
        Ima = ImageData.data;
        [filename, pathname, flag] = uigetfile('*.mat',...
            '请导入 ModulePca 算法压缩的图像矩阵');
        if ~flag
            return; % flag 为 0,用户取消选择,返回;
        end
        a = load([pathname filename]); % 载入存储的压缩图像
        CompressIma = a.CompressIma;
        ModulePca = getappdata(TrainImagesH,'ModulePca');
        Rebuilt = cellfun(@(x) ModulePca.EigV * x + ModulePca.Abar,CompressIma,...
            'UniformOutput',false); % 压缩的图像各个模块重建
        Rebuilt = cell2mat(Rebuilt); % 将重建的图像模块拼成图像
        figure('defaultuicontrolunits','normalized');
```

```
            NewAxesH = axes('Pos',[.08 .08 .72 ,0.72],'visible','on');
            imshow(uint8(Rebuilt)); % 显示重建后的图像
            title([get(get(AxesH,'title'),'string'),':用 ModulePca 算法重建后图像'])
            Mse = sqrt(mean(mean((Rebuilt - double(Ima)).^2 )) );
            uicontrol(gcf,'style','text','pos',[0.71 0.5 0.28,0.1],'string',...
                ['重建图像与原图像均方根误差:',num2str(Mse)],'fontsize',10);
        end

%=================================================================
% function ImaOut,导出主界面坐标轴中的图像,便于保存
%=================================================================
        function ImaOut(hObject,eventdata)
            NewFig = figure;
            NewAxesH = copyobj(AxesH,NewFig);
            set(NewAxesH,'position',[0.1300    0.1100    0.7750    0.8150]);
            set(NewFig,'colormap',get(fig,'colormap'));
        end
end
```

运行上述代码后会得到类似于图 8.1 所示界面，读者可以自行操作界面，在操作界面的过程中可以对源程序中感兴趣的代码行设置断点，从而方便了解程序的每一步的具体功能。

第 9 章

有关预测分类的案例

9.1 案例 16:北京市国民生产总值的灰色分析

9.1.1 概　述

在本案例里主要提出了一种基于数据融合的灰色预测方法。该方法具有较广泛的应用范围,将其运用到北京市国民生产总值 GNP 的预测中,取得了较好的预测效果。同时运用灰色系统理论中灰色关联度的理论分析了北京市国民生产总值同第一、第二、第三产业从业人数的关系紧密程度,从而给出了大力发展第二、第三产业理论的量化分析。

9.1.2 引　言

灰色系统理论是我国学者邓聚龙教授创立的,是近 20 年取得飞速发展的新兴理论。灰色系统理论专门针对贫信息、小样本数据进行分析,其特点是计算量小,对数据一般不需要假定服从某一特定的分布,而是直接从所面对的数据中提取信息。我国建国仅几十年,一些统计数据的数量极其有限,在分析这些数据时,灰色系统理论具有统计分析无法比拟的优势。灰色系统理论中的 $GM(1,1)$ 模型非常适合具有近指数规律数据的预测问题,所需数据量要求低,一般几个数据就可以建立起模型。

数据融合是目前研究的热门领域,其研究动力最初来源于军事领域。军事上为了精确分析获得的情报,以便从中获得尽可能多的信息,往往需要处理来自不同源头、不同类型甚至不同时期的数据。如何有效地从现有的数据进行融合进而获取比采用单一数据更加可靠的结果构成了数据融合的研究重点。

灰色关联度理论提供了衡量彼此有关系的两个时间数列之间关联程度的一种量化方法。通过对原始时间序列或者现有关联度理论的一定处理、变形,可以建立起不同的关联度,如灰色绝对关联度、灰色相对关联度、灰色综合关联度。

胡振涛,刘先省在 2005 年提出了一种针对多次测量所得数据的融合方法。本案例考虑到所用数据量的不同,建立的 $GM(1,1)$ 模型的预测结果也有相应的变化,故采用不同数据量的数据建立不同 $GM(1,1)$ 预测模型,得到不同的预测值,进而可用数据融合的方法对其进行融合生成最终预测值,对北京市 GNP 的预测结果表明该方法具有较好的预测精度。在预测北京市 GNP 的同时,本案例还分析了第一、第二、第三产业的从业人员与 GNP 的关联程度,定量分析了为什么要大力发展第二、第三产业。

9.1.3 灰色数据融合预测算法与灰色关联度

1. GM(1,1)模型的基本理论

定理 9.1 设 $X^{(0)}$ 为非负序列：

$$X^{(0)}=(x^{(0)}(1),x^{(0)}(2),\cdots,x^{(0)}(n)),x^{(0)}(k)\geqslant 0,\quad k=1,2,\cdots,n$$

$X^{(1)}$ 为 $X^{(0)}$ 的一阶累加生成(first order accumulated generating operation,1-AGO) 序列：

$$X^{(1)}=(x^{(1)}(1),x^{(1)}(2),\cdots,x^{(1)}(n)),\ x^{(1)}(k)=\sum_{i=1}^{k}x^{(0)}(i),\quad k=1,2,\cdots,n$$

$Z^{(1)}$ 为 $X^{(1)}$ 的紧邻均值生成序列，即

$$Z^{(1)}=(z^{(1)}(2),z^{(1)}(3),\cdots,z^{(1)}(n))$$

其中，$z^{(1)}(k)=0.5(x^{(1)}(k)+x^{(1)}(k-1))(k=2,3,\cdots,n)$。

若 $\hat{a}=[a,b]^{\mathrm{T}}$ 为参数列，且

$$\boldsymbol{Y}=[x^{(0)}(2),x^{(0)}(3),\cdots,x^{(0)}(n)]^{\mathrm{T}},\ \boldsymbol{B}=\begin{bmatrix}-z^{(1)}(2) & -z^{(1)}(3) & \cdots & -z^{(1)}(n)\\ 1, & 1, & \cdots & 1\end{bmatrix}^{\mathrm{T}}$$

则 $GM(1,1)$ 模型：

$$x^{(0)}(k)+az^{(1)}(k)=b$$

的最小二乘估计参数列满足：

$$\hat{a}=(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{Y}$$

若设 $\beta=\dfrac{b}{1+0.5a},\alpha=\dfrac{a}{1+0.5a}$，则

$$x^{(0)}(k)=(\beta-\alpha x^{(0)}(1))\mathrm{e}^{-a(k-2)}\tag{9-1}$$

式(9-1)就是本案例用以预测的依据方程式。详细证明可以参见文献[6]。

$GM(1,1)$ 模型的 MATLAB 代码如下：

```
function pre = GM11(x)
 % x,建模数据序列 x = [x(1),x(2),...,x(n)]
 % pre,由 GM11 模型得到的预测值,即由 x 估计出的 x(n+1)
n = length(x);
x1 = cumsum(x);
z1 = (x1(1:n-1) + x1(2:n))/2;
Y = x(2:length(x))';
B = [ - z1',ones(n-1,1)];
A = (B' * B)\B' * Y; % GM(1,1)模型的 a,b 参数组成的向量。
beta = A(2)/(1 + .5 * A(1));
alpha = A(1)/(1 + .5 * A(1));
 % 初始序列 x0(k)模拟后的随时间 t 变化的序列
f = @(t)(beta - (alpha) * x(1)) * exp( - (A(1) * (t - 2)));
pre = f(n + 1);
```

2. 灰色数据融合预测算法

设 $a_1,a_2,\cdots,a_m$ 为由 m 个 $GM(1,1)$ 模型预测得到的期望时刻的值。定义任意两个值之间的距离如下：

$$d_{ij}=|a_i-a_j|,\quad i,j\in\{1,2,\cdots,m\}$$

构造两个数据间支持度函数

$$r_{ij}=\cos\left(\frac{\pi d_{ij}}{2_{\max}\{d_{ij}\}}\right),\quad i,j\in\{1,2,\cdots,m\} \tag{9-2}$$

满足：

1) r_{ij} 与相对距离成反比关系，即两个值相差越大，彼此之间支持程度越小。

2) $r_{ij}\in[0,1]$，使数据的处理能够利用模糊集合理论中隶属函数的优点，避免数据之间相互支持程度的绝对化。

建立如下的支持度矩阵：

$$\boldsymbol{R}=(r_{ij})_{m\times m}$$

为了从 $a_1,a_2,\cdots,a_m$ 融合得到最终的 a，需要确定每个 a_i 的权重 ω_i，ω_i 满足 $\sum_{i=1}^{m}\omega_i=1$，注意到 ω_i 应综合包含 $r_{i1},r_{i2},\cdots,r_{im}$ 的信息，从而要寻找一组非负数 $v_1,v_2,\cdots,v_n$，使得

$$\omega_i=\sum_{j=1}^{m}v_j r_{ij} \tag{9-3}$$

式(9-3)可以改写为矩阵形式：

$$\boldsymbol{W}=\boldsymbol{RV}$$

其中，$\boldsymbol{W},\boldsymbol{V}$ 分别为由 ω_i 和 v_j 组成的列向量；$\boldsymbol{R}$ 为非负对称矩阵，存在最大模特征值 λ，与其对应的特征向量为

$$\boldsymbol{V}_\lambda=[v_1^\lambda,v_2^\lambda,\cdots,v_m^\lambda]$$

分量全为非负数，由特征向量特征值性质，可以取

$$\omega_i=\frac{v_i^\lambda}{\sum_{j=1}^{m}v_j^\lambda} \tag{9-4}$$

融合后得到

$$a=\sum_{i=1}^{m}\omega_i a_i \tag{9-5}$$

该算法的 MATLAB 代码如下：

```
function [a,w] = DataFusionS(x)
% 对于多次测量或者预测组成的 x 的观察值进行结果融合；
% x 为行向量。
% 返回值：a,融合结果；w,x 中每个分量在生成 a 的时候所占的权重
% 调用格式[a,w] = DataFusionS(x);
Lx = length(x);
switch Lx
    case 0
        a = 0;
        w = [];
    case 1
        a = x;
        w = 1;
    case 2
        a = mean(x);
```

```
        w = [1/2,1/2];
    otherwise
        DataCombine = nchoosek(1:length(x),2); % x元素两两之间可能的所有组合方式
        d = abs( x(DataCombine(:,1)) - x(DataCombine(:,2)));
        maxd = max(d);
        [Y,X] = meshgrid(1:Lx,1:Lx);
        R = cos(pi * ( x(X) - x(Y))/(2 * maxd)); % 构造支持度矩阵
        [V,D] = eig(R);
        w = V(:,Lx)/sum(V(:,Lx));
        a = x * w;
end
```

例如，在某个比赛项目中，一共有10个评委打分，给出的分数分别为9、9.5、9.7、8.5、10、9.0、9.2、9.3、9.5、9.6，按照通常的做法，我们会去掉一个最高分，去掉一个最低分，剩下的取平均值作为选手的最后得分。可是，严格来讲，如果专家比较权威，最高分最低分也反映了选手某方面的一些情况，贸然去掉的话会损失一些信息。如果采用上面的数据融合算法，可以充分利用这些信息，同时这些信息又不会对结果造成决定性影响。

利用上面的程序进行计分，代码如下：

```
x = [9  9.5  9.7  8.5  10  9.0  9.2  9.3  9.5  9.6];
[a,w] = DataFusionS(x)
>> x = [9  9.5  9.7  8.5  10  9.0  9.2  9.3  9.5  9.6];
>> [a,w] = DataFusionS(x)
a =
    9.3486
```

如果我们去掉一个最高分和最低分，然后取平均值，最后的得分是9.35分；直接求平均值，最后得分是9.33分；而用融合法求平均值，最后的得分是9.348 6分。由此可见，该融合方法比直接求平均值法具有较强的稳健性，尤其是有异常值的时候，它能参考异常值所包含的信息，同时也不至于过多受异常值的影响。

灰色关联度

篇幅所限，有关灰色绝对关联度以及灰色相对关联度的定义及详细计算公式参见文献[6]，这里只给出相应关联度的MATLAB计算代码。灰色相对关联度是对原始数列进行“初值化”（即原始数列每个元素都除以第一个元素）后的新数列的绝对关联度。为了多方面比较，在这个案例里提出了一种将原始数列先“归一化”，然后再算新数列绝对关联度的方法，称之为“灰色归一关联度”。

灰色绝对关联度的MATLAB代码如下：

```
function r = GreyAbsR(x,y)
% 计算序列 x,y 的灰色绝对关联度
% 调用语法 r = GreyAbsR(x,y)
if length(x) ~ = length(y)
    errordlg('两序列长度不等！')
    return
end
x0 = x - x(1);
```

```
y0 = y - y(1);
z0 = y0 - x0;
s0 = abs( sum( x0(2:end - 1)) + 0.5 * x0(end));
s1 = abs( sum( y0(2:end - 1)) + 0.5 * y0(end));
s10 = abs( sum( z0(2:end - 1)) + 0.5 * z0(end));
s = 1 + s0 + s1;
r = s/(s + s10);
```

灰色相对关联度的 MATLAB 代码如下：

```
function r = GreyRR(x,y)
% 计算序列 x,y 的灰色相对关联度,要求各序列的第一个元素不能为 0
% 调用语法 r = GreyRR(x,y)
if ~(x(1)&&y(1))
    errordlg('序列首元素不能为 0! ')
    return
end
x = x/x(1);
y = y/y(1);
r = GreyAbsR(x,y);
```

灰色归一关联度的 MATLAB 代码如下：

```
function r = GreyUniR(x,y)
% 计算序列 x,y 的灰色归一关联度
% 调用语法 r = GreyUniR(x,y)
xsize = size(x);
ysize = size(y);
if (min(xsize) ~ = 1)||(min(ysize) ~ = 1)
    errordlg('x 或 y 必须是向量! ');
    return
end
if xsize(2) == 1 % 若是列向量转化成行向量
    x = x';
end
if ysize(2) == 1
    y = y';
end
xyminmax = minmax([x;y]);
x = (x - xyminmax(1,1))/(xyminmax(1,2) - xyminmax(1,1)); % 归一化
y = (y - xyminmax(2,1))/(xyminmax(2,2) - xyminmax(2,1));
r = GreyAbsR(x,y);
```

9.1.4　实例分析

1. 灰色数据融合预测算法预测效果

从北京市统计局网站上下载了北京市 1987～2002 年的 GNP 以及第一、第二、第三产业的职工从业人数数据，见表 9－1。

表 9-1　北京市 1987～2002 年国民生产总值以及三大产业职工人数

年　份	国民生产总值/亿元	第一产业职工人数/万人	第二产业职工人数/万人	第三产业职工人数/万人
1987	326.82	91.8	264.1	242.3
1988	410.22	88.4	267.6	228.1
1989	455.86	91.0	266.3	236.6
1990	500.72	90.7	281.6	254.8
1991	598.79	90.8	279.7	263.5
1992	709.00	84.5	281.6	283.2
1993	863.23	65.1	279.4	283.3
1994	1 084.33	73.2	272.2	318.9
1995	1 395.12	70.6	271.0	323.7
1996	1 616.03	72.5	260.1	327.6
1997	1 810.49	71.0	257.6	328.1
1998	2 011.77	71.5	226.0	324.7
1999	2 174.97	74.5	216.2	327.9
2000	2 479.32	72.9	208.2	338.2
2001	2 846.29	71.2	215.9	341.8
2002	3 213.44	67.6	235.3	376.3

对于国民生产总值的预测，采用要预测年份之前的 4 个，5 个，…，11 个数据来分别建立 *GM*(1,1)模型进行预测。比如预测 2001 年的 GNP，我们采用 1997～2000 年，1996～2000 年，……，1990～2000 年的数据分别建模预测，得到 8 个预测值，然后对其进行融合得到最终的预测值。其代码如下：

```
GNP = [326.82  410.22  455.86  500.72  598.79  709.00  863.23  1084.33  1395.12  1616.03...
    1810.49  2011.77  2174.97  2479.32];
lGNP = length(GNP);
pre = arrayfun(@(k) GM11(GNP(lGNP - k:end)),3:10)

pre =

    1.0e + 003 *

    2.7372    2.7240    2.7286    2.7552    2.8275    2.9005    2.9645    3.0188
>> p = DataFusionS(pre)
p =
  2.8135e + 003
```

可以看出灰色融合预测给出的预测值为 2 813.5，与实际值相比误差为 1.15%。

同理，预测 2002 年的 GNP，用同样的办法可以得到 8 个预测值：3 252.2，3 178.1，3 144.8，3 132.0，3 144.0，3 203.3，3 270.6，3 334.6。最后融合的结果为 3 198.9，与实际值相

比误差为 0.45% 。从上面分析可见，该算法具有较强的实用性，虽然对预测结果直接求平均值似乎在某些情况下精确度更好，但是其稳健性较差，当数据有异常值时，往往效果不好。

2. 关联度分析

采用灰色绝对关联度、灰色相对关联度、以及"灰色归一关联度"方法对北京市 GNP 和各个产业从业人数进行关联分析，代码如下：

```
GNP = [326.82 410.22 455.86 500.72 598.79 709.00 863.23 1084.33 1395.12...
    1616.03 1810.49 2011.77 2174.97 2479.32 2846.29 3213.44];
m1 = [91.8 88.4 91.0 90.7 90.8 84.5 65.1 73.2 70.6 72.5 71.0 ...
    71.5 74.5 72.9 71.2 67.6];
m2 = [264.1 267.6 266.3 281.6 279.7 281.6 279.4 272.2 271.0 260.1 257.6 226.0...
    216.2 208.2 215.9 235.3 ];
m3 = [242.3 228.1 236.6 254.8 263.5 283.2 283.3 318.9 323.7 327.6 328.1 324.7...
    327.9 338.2 341.8 376.3 ];
r1 = GreyAbsR(GNP,m1); % GNP 和第一产业从业人数的灰色绝对关联度
r2 = GreyAbsR(GNP,m2); % GNP 和第二产业从业人数的灰色绝对关联度
r3 = GreyAbsR(GNP,m3); % GNP 和第三产业从业人数的灰色绝对关联度
r4 = GreyRR(GNP,m1); % GNP 和第一产业从业人数的灰色相对关联度
r5 = GreyRR(GNP,m2); % GNP 和第二产业从业人数的灰色相对关联度
r6 = GreyRR(GNP,m3); % GNP 和第三产业从业人数的灰色相对关联度
r7 = GreyUniR(GNP,m1); % GNP 和第一产业从业人数的灰色归一关联度
r8 = GreyUniR(GNP,m2); % GNP 和第二产业从业人数的灰色归一关联度
r9 = GreyUniR(GNP,m3); % GNP 和第三产业从业人数的灰色归一关联度
```

将得到的各个关联度总结到表 9-2 中。

表 9-2 北京市 1987～2002 年国民生产总值和各产业从业人数的各种灰色关联度对比

灰色关联度	GNP 和第一产业从业人数	GNP 和第二产业从业人数	GNP 和第三产业从业人数
灰色绝对关联度	0.500 0	0.500 0	0.527 0
灰色相对关联度	0.504 9	0.505 1	0.541 2
灰色归一关联度	0.518 1	0.532 3	0.977 0

从表 9-2 可以看到，各产业从业人数对 GNP 的影响情况为：第三产业≥第二产业≥第一产业。其中第三产业对 GNP 的影响最大，第一、第二产业对 GNP 的影响差别不是很大。这样就运用灰色关联度理论给出了为什么大力发展第二、第三产业的量化说明，其方法以及结果可供有关部门借鉴。

本案例演示了一种基于数据融合的灰色预测方法。该方法具有较广的应用范围。为了说明这一点，案例预测了北京市的 GNP，并且取得了较好的预测效果。同时运用灰色关联度理论给出了一个为什么要大力发展第二、第三产业的量化说明。案例中基于数据融合的灰色预测方法还有许多值得研究的地方，如支持度函数的选择、基本预测方法的选择等，感兴趣的读者可以自己去尝试。在灰色关联度理论方面的研究也可以尝试新的"灰色关联度"。

9.2 案例17：距离判别法与Bayes判别法在分类中的应用

9.2.1 概 述

判别分析是利用已知类别的样本训练模型，为未知样本判别的一种统计方法。判别分析产生于20世纪30年代，近年来，在自然科学、社会学以及经济管理学科中都有广泛的应用。

判别分析的特点是根据已掌握的、历史上每个类别的若干样本的数据信息，总结出客观事物分类的规律性，建立判别公式和判别准则。当遇到新的样本点时，只要根据总结出来的判别公式和判别准则，就能判别该样本点所属的类别。常用的判别分析方法有距离判别法、Bayes判别法、Fisher判别法和逐步判别法等。

判别在某种意义上讲也可以被认为是分类。本案例以Bayes判别法为例展示了Bayes判别法在分类中的应用。其中，距离判别法的基本思想是：样品和哪个总体距离最近，就判断它属于哪个总体，因此距离判别法也称为直观判别法。距离判别法只要求知道总体的特征量（即参数）——均值和协方差阵，不涉及总体的分布类型，当参数未知时，就用样本均值和样本协方差阵来估计。所以，距离判别法简单、结论明确，比较实用。其缺点是，该判别法与各总体出现的机会大小（即先验概率）完全无关，并且没有考虑错判造成的损失，这是不合理的。

Bayes判别法刚好解决了上述提到的问题。Bayes的统计思想总是假定对研究的对象已有一定的认识，常用先验概率分布来描述这种认识，然后抽取一个样本，用样本来修正已有的认识（先验概率分布），得到后验概率分布。各种统计推断都通过后验概率分布来进行。将Bayes思想用于判别分析就得到了Bayes判别法。

关于两种判别法的详细介绍可以参考文献[11]，限于篇幅这里对其理论不进行过多描述。

9.2.2 判别方法GUI

本小节给出这两种判别方法实现的简单的GUI界面，为方便读者操作，准备了一些测试数据，见北京航空航天大学出版社网站或MATLAB中文论坛相关在线交流版块。train是训练样本（判别准备前用的）；test是测试样本，即新数据，用来判别新样本中每一个属于哪一类的。这里给的例子属性个数$n=3$，实际使用时，n可以不局限于3。训练样本只要按照附件中的格式（即第一列为类名，其余列为属性）存为扩展名为.xls文件即可。测试样本是直接由属性列组成的，每一行表示一个样本。源代码如下：

```
function DiscriminantMethods
fig = figure('defaultuicontrolunits','normalized','name','判别方法比较',...
    'numbertitle','off','menubar','none'); % 主界面，返回主界面句柄 fig
UiButtonGroupH = uibuttongroup('Position',[0.55 0.08 0.40 0.85],'title',...
    '判别方法','fontsize',12,'fontname','隶书','units','normalized',...
    'backgroundcolor',[0.9 0.8 0.7],'bordertype','etchedin',...
    'ForegroundColor',[1 0 0],'ShadowColor',[0 1 1]); % 群组对象，并返回句柄
DistanceH = uicontrol('Style','Radio','String','距离判别法','fontsize',12,...
    'pos',[0.05  0.73  0.9  0.15],'backgroundcolor',[0.9  0.8  0.7],...
    'parent',UiButtonGroupH); % 距离判别法的选项
```

```
BayesH = uicontrol('Style','Radio','String','Bayes 判别法','fontsize',12,...
    'pos',[0.05  0.32  0.9  0.15],'backgroundcolor',[0.9  0.8  0.7],...
    'parent',UiButtonGroupH); % Bayes 判别法的选项
% 下面几行建立相关按钮控件。各菜单以及按钮单击时的回调函数(写在下面)
PrepareH = uicontrol(fig,'style','pushbutton','Pos',[0.15,0.68,0.2,0.08],...
    'string','判别前准备','fontsize',12,'callback',@Prepare);
StartDiscriminantH = uicontrol(fig,'style','pushbutton','Pos',...
    [0.15,0.38,0.2,0.08],'string','开始判别','fontsize',12,...
    'callback',@StartDiscriminant);
%==================================================
% function:Prepare,判别前准备按钮的回调函数
%==================================================
    function Prepare(hObject,eventdata)
        [filename, pathname, flag] = uigetfile('*.xls',...
            '当前选定的判别法的训练样本数据(*.xls)','*.xls'); % 弹出提示对话框
        if ~flag
            return; % flag 为 0,用户取消选择,返回;
        end
        [feature,class] = xlsread([pathname,filename]); % 读入训练样本数据
        lf = size(feature,1); % 训练样本个数
        Uclass = unique(class); % 得出训练样本的类别信息(即分属于哪几类)
        lUc = length(Uclass); % 类别数目
        Cfeature = cell(lUc,1);
        for k = 1:lUc
            % 训练样本中每一类特征归到一起
            Cfeature{k} = feature(strcmp(class,Uclass{k}),:);
        end
        WhichDisc = get(findobj(UiButtonGroupH,'style','radio','value',1),...
            'string'); % 当前界面中选择的是哪种判别法
        switch WhichDisc
            case '距离判别法'
                % 用来存储每一类的各特征平均值
                MeanCfeature = zeros(lUc,size(Cfeature{1},2));
                InvSigma = cell(lUc,1); % 用来存储每一类的协方差矩阵的逆
                for ii = 1:lUc
                    % 得到每一类的各特征平均值
                    MeanCfeature(ii,:) = mean(Cfeature{ii});
                    % 得到每一类的协方差矩阵的逆
                    InvSigma{ii} = inv(cov(Cfeature{ii}));
                end
                SampleInfo.MeanCfeature = MeanCfeature;
                SampleInfo.InvSigma = InvSigma;
                SampleInfo.Uclass = Uclass;
                setappdata(DistanceH,'SampleInfo',SampleInfo);
            case 'Bayes 判别法'
                % 用来存储每一类的各特征平均值
                MeanCfeature = zeros(lUc,size(Cfeature{1},2));
                Sigma = cell(lUc,1); % 用来存储每一类的协方差矩阵
                InvSigma = cell(lUc,1); % 用来存储每一类的协方差矩阵的逆
                PrioP = zeros(lUc); % 用来存储先验概率
```

```
                for ii = 1:lUc
                    % 得到每一类的各特征平均值
                    MeanCfeature(ii,:) = mean(Cfeature{ii});
                    Sigma{ii} = cov(Cfeature{ii}); % 得到每一类的协方差矩阵
                    InvSigma{ii} = inv(Sigma{ii}); % 得到每一类的协方差矩阵的逆
                    PrioP(ii) = length(Cfeature{ii})/lf;
                end
                SampleInfo.MeanCfeature = MeanCfeature;
                SampleInfo.Sigma = Sigma;
                SampleInfo.InvSigma = InvSigma;
                SampleInfo.Uclass = Uclass;
                SampleInfo.PrioP = PrioP;
                setappdata(BayesH,'SampleInfo',SampleInfo);
            otherwise
        end
        h = msgbox('准备完毕，请单击开始判别进行测试！'); % 弹出对话框
        uiwait(h); % 等待用户确认后再运行程序
    end
%============================================================
% function:StartDiscriminant,开始判别按钮的回调函数
%============================================================
    function StartDiscriminant(hObject,eventdata)
        [filename, pathname, flag] = uigetfile('*.xls',...
            '请导入测试样本数据(*.xls)','*.xls'); % 弹出提示对话框
        if ~flag
            return; % flag为0,用户取消选择,返回;
        end
        feature = xlsread([pathname,filename]);
        lf = size(feature,1);
        WhichDisc = get(findobj(UiButtonGroupH,'style','radio','value',1),...
            'string'); % 当前界面中选择的是哪种判别法
        switch WhichDisc
            case '距离判别法'
                % 获取判别前准备完成时存储的用于判别的信息
                SampleInfo = getappdata(DistanceH,'SampleInfo');
                if isempty(SampleInfo) % 如果SampleInfo为空值
                    hstr = '距离判别法还没有准备,请单击判别前准备按钮做相应准备！';
                    h = msgbox(hstr);
                    uiwait(h);
                    return;
                end
                lUc = length(SampleInfo.Uclass); % 类别数目
                % 存储每个样本到每个类的距离,行对应样本,列对应类
                Dist = zeros(lf,lUc);
                for k = 1:lf
                    for kk = 1:lUc
                      Dist(k,kk) = (feature(k,:) - SampleInfo.MeanCfeature(kk,:)) * ...
                        SampleInfo.InvSigma{kk} * (feature(k,:) - ...
                        SampleInfo.MeanCfeature(kk,:))';
```

```
            end % 计算每个样本到每个类的距离
        end
        [md,ClassNum] = min(Dist,[],2);
        Class = SampleInfo.Uclass(ClassNum);
        uiwait(helpdlg('当前样本已用距离判别法判别完毕,请按提示保存结果',...
            '距离判别法判别'));
        [file,path] = uiputfile([filename,'距离判别法判别结果.xls'],...
            '保存当前判别结果为:');
        if ~file
            uiwait(msgbox('用户放弃保存!'));
            return;
        end
        xlswrite([path,file],Class);
    case 'Bayes 判别法'
        % 获取判别前准备完成时存储的用于判别的信息
        SampleInfo = getappdata(BayesH,'SampleInfo');
        if isempty(SampleInfo) % 如果 SampleInfo 为空值
            hstr = 'Bayes 判别法还没有准备,请单击判别前准备按钮做相应准备!';
            h = msgbox();
            uiwait(h);
            return;
        end
        lUc = length(SampleInfo.Uclass); % 类别数目
        % 存储每个样本到每一类的判别函数,行对应样本,列对应类
        DiscrimFun = zeros(lf,lUc);
        for k = 1:lf
            for kk = 1:lUc
                DiscrimFun(k,kk) = ( 2 * log(SampleInfo.PrioP(kk)) - ...
                    log(det(SampleInfo.Sigma{kk})) - ...
                    (feature(k,:) - SampleInfo.MeanCfeature(kk,:)) * ...
                    SampleInfo.InvSigma{kk} * ...
                    (feature(k,:) - SampleInfo.MeanCfeature(kk,:))')/2;
            end % 计算每个样本到每个类的判别函数
        end
        [md,ClassNum] = max(DiscrimFun,[],2);
        Class = SampleInfo.Uclass(ClassNum);
        uiwait(helpdlg('当前样本已用 Bayes 判别法判别完毕,请按提示保存结果',...
            'Bayes 判别法判别'));
        [file,path] = uiputfile([filename,'Bayes 判别法判别结果.xls'],...
            '保存当前判别结果为:');
        if ~file
            uiwait(msgbox('用户放弃保存!'));
            return;
        end
        xlswrite([path,file],Class);
    otherwise
    end
  end
end
```

运行上面代码得到如图9.1所示界面。

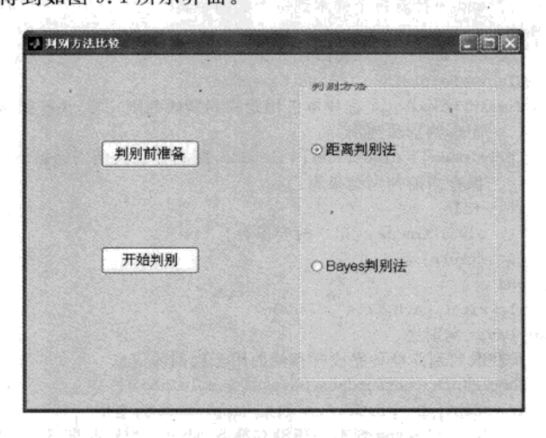


图9.1　距离判别和Bayes判别界面

9.2.3　判别方法GUI应用举例

上述GUI可以用来实现分类，操作很简单，读者可以根据提示自己进行操作。本书免费附赠的源文件中的train.xls和test.xls分别是训练模型和测试模型文件。train文件里面的每一行表示一个样本，其中第一列是所属类名。关于类名并无特别的格式要求，只是要求其为字符形式，并且属于同一类的名称必须要一致。

采用Bayes判别法对test中的数据进行判别后，会生成一个Excel结果文档。根据结果文档中划分的类别归属将test文件中的数据和train文件中的数据合并到一起，如图9.2所示。

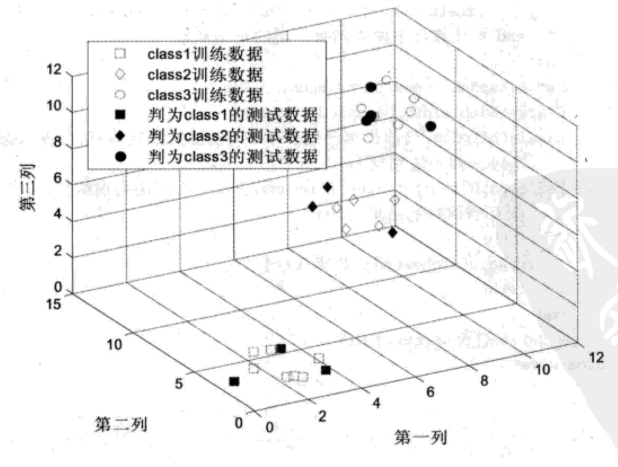


图9.2　训练样本和Bayes判别法判别后的测试样本

从图 9.2 可以看出，Bayes 判别法对测试数据进行了准确的判别。读者也可以按照 train 文件和 test 文件格式自行准备训练样本和测试样本，每个样本的属性不局限于 3，并且类的名称可以随意，只要是字符型并且同一类的名称唯一即可。

9.3 案例 18：支持向量机的应用

9.3.1 概　述

支持向量机（Support Vector Machine，SVM）是 Vapnik 教授等人在多年研究统计学理论基础上提出的一种用于解决线性不可分这样的分类问题的理论。支持向量机的基本思想是：定义最优线性超平面，并把寻找最优线性超平面的算法归结为求解一个凸规划问题。进而基于 Mercer 核展开定理，通过构造一个非线性映射 φ，把样本空间映射到一个高维乃至无穷维的特征空间，使之在特征空间中可以应用线性学习机的方法解决样本空间中的高度非线性分类和回归等问题。

支持向量机方法是一种具有严密理论基础的计算机学习的新方法，它已经成为计算机学习、模式识别、计算智能、预测预报等领域的热点技术，在国内外受到广泛关注。本案例将简要介绍支持向量机理论、MATLAB 自带的有关支持向量机的函数以及展示支持向量机的一些应用案例。

9.3.2 支持向量机介绍

1. 两分类支持向量机

这里介绍本案例用到的两分类支持向量机，即 C-支持向量机，具体形式如下：

1）设已知训练集：

$$T=\{(\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_l,y_l)\}\in(X\times Y)^l$$

其中，$\boldsymbol{x}_i\in X=R^n, y_i\in Y=\{1,-1\}(i=1,2,\cdots,l)$；$\boldsymbol{x}_i$ 为特征向量。

2）选取适当的核函数 $K(\boldsymbol{x},\boldsymbol{x}')$ 和适当的参数 C，构造并求解最优化问题：

$$\min_{\alpha}\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j\alpha_i\alpha_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)-\sum_{j=1}^{l}\alpha_j$$

$$\text{s.t.}\quad \sum_{i=1}^{l}y_i\alpha_i=0,$$

$$0\leqslant\alpha_i\leqslant C,\quad i=1,\cdots,l$$

得最优解：$\alpha^*=(\alpha_1^*,\cdots,\alpha_l^*)^{\mathrm{T}}$。

3）选取 α^* 的一个正分量 $0<\alpha_j^*<C$，并据此计算阈值：

$$b^*=y_j-\sum_{i=1}^{l}y_i\alpha_i^*K(\boldsymbol{x}_i,\boldsymbol{x}_j)$$

4）构造决策函数

$$f(\boldsymbol{x})=\mathrm{sgn}\left(\sum_{i=1}^{l}\alpha_i^*y_iK(\boldsymbol{x},\boldsymbol{x}_i)+b^*\right)$$

2. 多分类支持向量机

直接应用上面的两分类支持向量机只能解决两分类问题，现实中面临的很多问题都是多

分类问题，这时为了解决多分类问题就需要构造一系列的两分类支持向量机。这里将要介绍两种构造一系列两分类支持向量机的方法。

设已知训练集：

$$T=\{(\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_l,y_l)\}\in(X\times Y)^l$$

其中：

$$\boldsymbol{x}_i\in X=R^n,\quad y_i\in Y=\{1,2,\cdots,M\},i=1,2,\cdots,l$$

首先对所有的

$$(i,j)\in\{(i,j)\mid i\leqslant j,i,j=1,2,\cdots,M\}$$

进行下列运算：在训练集中抽取所有 $y=i$ 和 $y=j$ 的样本点。基于这些样本点组成一个训练集 T_{i-j}。用求解两类分类问题的支持向量机求得实值函数 $g^{i-j}(x)$ 和判定 $x\in X$ 属于第 i 类或第 j 类的分类机：

$$f^{i-j}(\boldsymbol{x})=\begin{cases}i, & g^{i-j}(x)>0\\ j, & 其他\end{cases}$$

然后在需要对给定的一个测试输入 $\boldsymbol{x}$ 推断它属于 M 类中的第几类时，考虑上述所有的分类机对 $\boldsymbol{x}$ 所属类别的意见：一个分类机判定 $\boldsymbol{x}$ 属于第 i 类就意味着第 i 类获得一票。得票数最多的类别就是最终判定 $\boldsymbol{x}$ 所属的类别。

另外一种方法是一类对余类构造支持向量机的方法。对于训练集：

$$T=\{(\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_l,y_l)\}\in(X\times Y)^l$$

其中：

$$\boldsymbol{x}_i\in X=R^n,\quad y_i\in Y=\{1,\cdots,M\},\quad i=1,\cdots,l$$

在训练支持向量机时，可以将所有 y_i 相等的归为一类，剩下的归为一类，这样训练好的支持向量机就可以识别 y_i 类了。按照上述方法一共可以构造 M 个支持向量机，其中每个支持向量机负责识别相应的第 y_i 类。

9.3.3 MATLAB所依据的支持向量机模型

MATLAB 2008a 中和支持向量机分类有关的函数主要是 svmtrain 和 svmclassify，前者用于训练支持向量机，后者用训练好的支持向量机来分类。最关键的是 svmtrain 函数，其不同的参数输入直接决定了最终生成的支持向量机的结构。下面介绍 svmtrain 函数的结构。

1. svmtrain 函数结构

svmtrain 函数所依据的是如下 C-支持向量机的变形。

1) 设已知训练集：

$$T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_l,y_l)\}\in(X*Y)^l$$

其中，$\boldsymbol{x}_i\in X=R^n,y_i\in\{1,-1\}(i=1,2,\cdots,l)$

2) 选取适当的核函数 $K(x,x')$ 和适当的参数 C，构造并求解最优化问题

$$\min_{\alpha}\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j\alpha_i\alpha_j\left(K(\boldsymbol{x}_i,\boldsymbol{x}_j)+\frac{1}{C}\delta_{ij}\right)-\sum_{j=1}^{l}\alpha_j \tag{9-6}$$

$$\text{s.t.}\ \sum_{i=1}^{l}y_i\alpha_i=0$$

$$\alpha_i\geqslant0,\quad i=1,2,\cdots,l$$

其中：

$$\delta_{ij}=\begin{cases}1, & i=j\\0, & i\neq j\end{cases}$$

得最优解 $\alpha^*=(\alpha_1^*,\cdots,\alpha_l^*)^T$；

3）选取 α^* 的一个正分量 $\alpha_j^*>0$，并据此计算阈值：

$$b^*=y_j\left(1-\frac{\alpha_j^*}{C}\right)-\sum_{i=1}^{l}y_i\alpha_i^*K(\boldsymbol{x}_i,\boldsymbol{x}_j)$$

4）构造决策函数：$f(\boldsymbol{x})=\mathrm{sgn}\left(\sum_{i=1}^{l}\alpha_i^*y_iK(\boldsymbol{x},\boldsymbol{x}_i)+b^*\right)$

2. svmtrain 函数所支持的核函数类型

关于核函数 $K(\boldsymbol{x},\boldsymbol{x}')$，svmtrain 函数中可以设定的核函数分为现有的核函数和用户自己定义的核函数两大类。现有的核函数有线性核函数（Linear kernel）、二次核函数（Quadratic kernel）、高斯径向基核函数（RBF kernel）、多项式核函数（Polynomial kernel）和多层感知器核函数（Multilayer Perceptron kernel）。

用户也可以自定义核函数，提前写好核函数文件，将其函数句柄传入 svmtrain 函数即可。

在案例中，核函数选取的是二次多项式核函数，具体形式为：

$$K(\boldsymbol{u},\boldsymbol{v})=(uv')(1+uv')$$

上面所列的核函数的函数文件可以在 matlabroot\toolbox\bioinfo\biolearning\private 目录下找到。

问题(9-6)是一个二次规划问题。svmtrain 函数根据输入参数的不同可以用以下三种方式求解。

1）二次规划（Quadratic Programming）这个需要用户安装优化工具箱。如果用户安装了优化工具箱，那么默认的求解问题(9-6)就是该方法，即调用 quadprog 函数。

2）序列最小最优化（Sequential Minimal Optimization）。该方法适合于求解大型问题，同时用户未装优化工具箱时是默认的方法。

3）最小二乘法（Least-Squares）。

9.3.4 支持向量机实现图像分割

如图 9.3 所示，蓝色的湖面上一只黄色的小鸭子在捕食昆虫，该图片文件名为 littleduck.jpg，利用支持向量机，将小鸭子从背景中分割出来。

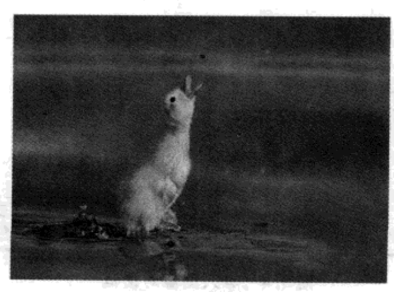


图 9.3　示例图像

利用支持向量机分类，首先需要找到区分不同像素点的特征。由于这幅图片的颜色对比鲜明，因此可以选取像素点的RGB值作为特征。偏黄色的点作为鸭子的特征点，偏蓝色的点作为湖水的特征点，另外考虑到鸭子的眼睛是黑色的，因此也可以适当选取一两个黑色的像素点作为鸭子的特征。

下面代码展示了读取图像，训练支持向量机并分割图像的一系列过程：

```
function DuckLakeSegmentation
% 导入图像文件引导对话框
[filename,pathname,flag] = uigetfile('*.jpg','请导入图像文件');
Duck = imread([pathname,filename]);
LakeTrainData = [52  74  87;76  117  150;19  48  62;35  64  82;46  58  36;...
    50  57  23;110  127  135;156  173  189;246  242  232;...
    166  174  151]; % 从图上选取的几个位于湖面区域的有代表的点的 RGB 值
% 从图上选取的几个位于鸭子区域的有代表的点的 RGB 值
DuckTrainData = [211  192  107;202  193  164;32  25  0;213  201  151;115  75  16;...
    101  70  0;169  131  22;150  133  87];
% 属于湖的点为 0,鸭子为 1
group = [zeros(size(LakeTrainData,1),1);ones(size(DuckTrainData,1),1)];
LakeDuckSVM = svmtrain([LakeTrainData;DuckTrainData],group,...
    'Kernel_Function','polynomial','Polyorder',2); % 训练得到支持向量分类机
[m,n,k] = size(Duck);
Duck1 = double(reshape(Duck, m * n, k)); % 将数组 Duck 转成 m * n 行,3 列的双精度矩阵
% 根据训练得到的支持向量机对整个图像的像素点分类
IndDuck = svmclassify(LakeDuckSVM,Duck1);
IndLake = ~IndDuck; % 属于湖的点的逻辑数组
result = reshape([IndLake, IndLake, IndLake],[m,n,k]); % 和图片的维数相对应
Duck2 = Duck;
Duck2(result) = 0;
figure
imshow(Duck2)
```

运行上述代码首先出现如图9.4所示的对话框。

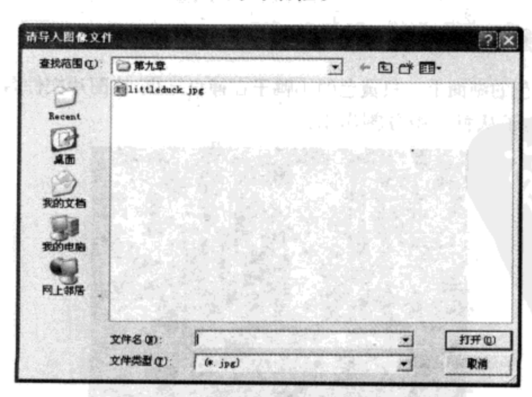


图9.4 导入图像对话框



选中图片 littleduck.jpg 后，程序即读取图像数据，并根据设置的训练点训练支持向量机，训练好的支持向量机为 LakeDuckSVM，它会把属于湖面区域的点归为"0"这一类，属于鸭子区域的点归为"1"这一类。接着 svmclassify 函数完成对图像剩余像素点的分类工作，通过逻辑数组 result 将原始图像数组中属于湖面的像素点的灰度值设置为 0，此时背景成了黑色，而鸭子区域的图像保持原样，这样就可以清楚地看到图像分割的效果(见图 9.5)。

需要说明的是，选取训练点时有一个技巧，就是刚开始先选取比较少的有代表性的几个点作为训练点，比如在鸭子区域选取几个有代表性的点，把它们的 RGB 值作为鸭子的训练样本，类似的操作可以得到湖面区域的训练样本。查看图片上某点的 RGB 值时可以利用图片 RGB 提取工具。这种绿色免费的软件在互联网上有很多，如 ColorPix 小工具。ColorPix 是 ColorSchemer 公司出品的一款绿色、免费小软件，整个程序只需要一个 EXE 可执行文件，操作非常方便。

最初选取的训练点可能导致图片分割得不是很好，例如鸭子后面溅起的水花被分割成鸭子区域了，这时候可以对错分的水花区域添加一两个典型点到湖面区域。一般尝试几次后就可以完全将鸭子图像从整个图像中分割出来。利用 DuckLakeSegmentation 程序中的那些训练点就可以达到图 9.5 所示的分割效果。



图 9.5　分割出鸭子的图像

9.3.5　支持向量机实现手写体数字识别

手写体数字的识别在社会经济生活的许多方面都有着广泛的应用。其识别方法也有许多种，如神经网络方法、Bayes 判别方法等。本案例将介绍支持向量机方法用于手写体数字的识别。

1. 训练支持向量机

本案例共选取 50 幅手写体数字图片作为训练样本，每个数字均有 5 幅图片，50 幅图片合在一起，如图 9.6 所示。

0 0 0 0 0 1 1 1 1 1
2 2 2 2 2 3 3 3 3 3
4 4 4 4 4 5 5 5 5 5
6 6 6 6 6 7 7 7 7 7
8 8 8 8 8 9 9 9 9 9

图 9.6　训练样本图片

初始训练图片中每个字符区域大小不一，而且不是二值图像，需要进行标准化操作，即对每幅图片进行反色后二值化处理，并截取包含完整数字的最小区域，并将该区域转化成标准的16×16像素大小的图像。转化后的图像是黑底白字的图，图中数字区域像素值为1，背景区域为0。例如对于第五个训练图像“8”，原始图像以及反色图像、二值化后图像、选取包含完整“8”的最小区域图像以及转化成16×16像素大小的图像，如图9.7所示。所有的训练图像标准化后，就可以构造并训练得到一系列支持向量机，这里采用9.3.2节介绍的第一种方法构造支持向量机，来依次完成0～9这10个数字的识别。

图9.7　图片标准化处理示例

训练图片标准化并构造、训练支持向量机的代码如下：

```
function SVM = StdPictureAndGenerateSVM
PicFormat = {'*.jpg','JPEG image (*.jpg)';...
    '*.bmp','Bitmap image (*.bmp)';...
    '*.*','All Files (*.*)'};
% 利用 uigetfile 函数交互式选取训练样本图片
[FileName FilePath,flag] = uigetfile(PicFormat,'导入原始训练图片',...
    '*.jpg','MultiSelect','on');
if ~flag
    return; % flag 为 0,用户取消选择,返回;
end
%将导入的图像样本名称用 cell 数组存储,因为导入一个图像时 filename 用 char 型存储,
%cellstr 作用后变为 filename 用 1*1 的 cell 数组存储。导入多个图像时 filename 用 cell 存储,
%cellstr 作用后不变
FileName = cellstr(FileName);
n = length(FileName);
TotalTraining = zeros(n,256); % 存储所有的训练样本
group = zeros(n,1); % 对应的组
for k = 1:n
    I = imread([FilePath,FileName{k}]); % 读入一幅图片
    I = 255 - I; % 图像反色处理
    I = im2bw(I,0.4); % 设定阈值 0.4,把反色后图像转成二值图像
    [y,x] = find(I == 1); % 查找数字上所有像素点的行标 y 和列标 x
    BW = I(min(y):max(y),min(x):max(x)); % 截取包含完成数字的最小区域图像
    %将截取的包含完整数字的最小区域图像转成 16×16 的标准化图像
    BW = imresize(BW,[16, 16]);
    % 将标准化图像按列拉成一个向量后并转置,生成 50×256 的训练样本矩阵 training
    TotalTraining(k,:) = double(BW(:)');
    group(k) = str2double(FileName{k}(4));
end
%第 k 个数字的训练图像存在 Training{k}里
Training = arrayfun(@(k) TotalTraining(group == k,:),0:9,'UniformOutput',false);
nchk092 = nchoosek(0:9,2); %0 到 9 数字两两配对
```

```
SVM = cell(size(nchk092,1),1); % 存储 45 个支持向量机,分别实现两两数字之间的比较
for k = 1:size(nchk092,1)
    t1 = Training{nchk092(k,1) + 1};
    t2 = Training{nchk092(k,2) + 1};
    % 在这个问题中,训练样本较少,多项式的 Polyorder 不宜过高,读者可以修改一下
    % Polyorder 的值,如改成 2 或 3,会发现识别准确率会大幅下降
    SVM{k} = svmtrain([t1;t2],[ones(size(t1,1),1);zeros(size(t2,1),1)],...
        'Kernel_Function','polynomial','Polyorder',1);
end
```

2. 测试支持向量机识别能力

另外选取 30 幅手写体数字图片作为测试样本,每个数字对应三幅测试图片,每幅图片大小均为 50×50 像素,文件名依次为 num0_1.bmp,num0_2.bmp,num0_3.bmp,…,num9_3.bmp。30 幅图片如图 9.8 所示。

0123456789
0123456789
0123456789

图 9.8　测试图片示例

在应用支持向量机对测试图像进行识别的时候,也要对其进行标准化处理,处理方法同训练图片的处理方法相同。测试支持向量机识别数字的代码如下:

```
function [result,percent,TrueNum] = SvmNumIdentify(SVM)
% SVM,训练好的 10 个支持向量机
% result,每个图片对应识别结果;percent,识别准确率
PicFormat = {'*.jpg','JPEG image (*.jpg)';...
    '*.bmp','Bitmap image (*.bmp)';...
    '*.*','All Files (*.*)'};
[FileName FilePath,flag] = uigetfile(PicFormat,'导入测试图片',...
    '*.bmp','MultiSelect','on');
if ~flag
    return;
end
FileName = cellstr(FileName);
n = length(FileName);
TrueNum = zeros(1,n); % 每幅测试图片代表的真实数字
result = zeros(n,1);
svmresult = zeros(length(SVM),1);
nchk092 = nchoosek(0:9,2);
for k = 1:n
    TrueNum(k) = str2double(FileName{k}(4)); % 每幅图片对应的真实数字
    I = imread([FilePath,FileName{k}]);
    I = 255 - I;
    I = im2bw(I,0.4);
    [y,x] = find(I == 1);
```

```
        BW = I(min(y):max(y),min(x):max(x));
        BW = imresize(BW,[16, 16]);
        Test = double(BW(:)');
        % 对测试图像依次用 length(SVM)个支持向量机识别
        for kk = 1:length(SVM)
            svmresult(kk) = svmclassify(SVM{kk},Test);
        end
        tmp = nchk092(:,1). * svmresult + nchk092(:,2). * ~svmresult;
        result(k) = mode(tmp);
    end
    result = result';
    percent = sum(result == TrueNum)/n;
```

运行从 StdPictureAndGenerateSVM 函数得到训练好的 SVM，然后再运行 SvmNumIdentify 函数，根据提示选择要测试的 30 个图像，结果如下：

```
    [result,percent,TrueNum] = SvmNumIdentify(SVM)
    result =
      Columns 1 through 16
        0     0     0     1     1     1     2     2     2     3     3     3     4     4     4     5
      Columns 17 through 30
        5     5     6     5     6     7     7     7     7     8     5     9     9     9
    percent =
        0.9000
    TrueNum =
      Columns 1 through 16
        0     0     0     1     1     1     2     2     2     3     3     3     4     4     4     5
      Columns 17 through 30
        5     5     6     6     6     7     7     7     8     8     8     9     9     9
```

result 是识别出的结果，TrueNum 是真实的结果。可见总共有三个图片出现了误判：一个是“6”被误判成“5”，还有一个“8”被误判成“7”，一个“8”被误判成“5”。在每个数字只有五个训练样本情况下，这样的判别效果还是可以接受的。读者在其他领域应用支持向量机解决分类问题时，可以尝试保证足够的训练样本个数，以降低误判比例。注意一个原则：训练样本比较少时，不要采用复杂的核函数；采用多项式核函数时，多项式的阶次不宜过高。

第10章 常微分方程(组)求解案例

10.1 案例19:常微分方程(组)解析求解案例

10.1.1 概 述

一些形式不是特别复杂的微分方程(组)可以求得其解析解,对于这类微分方程(组),可以利用MATLAB提供的符号计算函数dsolve很方便的求解。一般来说,相对于数值求解微分方程的一系列函数,dsolve函数调用格式要简洁、容易得多,这也是一些初学者喜欢使用dsolve函数求解微分方程的原因之一。

MATLAB从R2008b版本开始采用Mupad的符号计算内核,因此R2008b之前版本的dsolve函数和之后版本的dsolve函数在语法上以及返回结果上有些差异。一般来说,这些差异并不影响用户使用,但在有些时候需要注意。

10.1.2 dsolve函数

1. R2008b之前的dsolve函数

MATLAB R2008a是MATLAB采用Maple符号计算内核的最后一个版本,我们以这个版本的dsolve函数为例介绍R2008b的dsolve函数的用法。该版本的dsolve函数的语法规则如下:

```
r = dsolve('eq1,eq2,...', 'cond1,cond2,...', 'v')
r = dsolve('eq1','eq2',...,'cond1','cond2',...,'v')
```

r为求解得到的输出结果,eq1和eq2等是求解的微分方程表达式,微分方程表达式中自变量(以t为例)的n阶导数$\frac{d^n f}{dt^n}$可以用Dnf来表示,类似的,$\frac{d^n g}{dt^n}$可以用Dng来表示,等等。边界条件或者初值条件等一些微分方程的定解条件由cond1,cond2等给出。v为方程的自变量,默认的自变量是t。

在使用上述dsolve函数的第二种调用格式时,需要注意输入变量的个数不要超过12个。也就是说所有的方程、定解条件以及自变量总和不要超过12个。

2. R2008b以及以后版本的dsolve函数

R2008b之后版本的dsolve函数除了支持上述两种调用格式外,还多了一个IgnoreAnalyticConstraints设置项,其调用格式如下:

```
dsolve('eq1','eq2',...,'cond1','cond2',...,'v','IgnoreAnalyticConstraints',value)
```

其中IgnoreAnalyticConstraints参数项的字面意思是“忽略分析上的约束”，这是出于一些求解结果在一般性上的考虑。例如表达式 $\ln e^x$，通常我们认为其等于 x，这是基于默认 x 为实数情况下得出的结论。如果 $x=2\pi i$，则 $\ln e^{2\pi i}=\ln 1=0$ 而不是等于 $2\pi i$。上述IgnoreAnalyticConstraints有两个设置值可供选择：all和none。默认情况下为all，这个时候意味着不对所求结果进行一般意义上的推广，所以解出来的解可能在最一般意义条件下会不成立，但还是会满足原始微分方程以及定解条件。而如果选择none，dsolve返回的解（前提是能够求得解析解）在最一般意义下也会成立，但是会增加求不出统一的解析表达式的概率。

下面来看用dsolve函数求解微分方程的一些实例。

10.1.3 dsolve函数求解实例

【例10.1-1】 求 $\frac{dx}{dt}=y,\frac{dy}{dt}=-x$ 的解。本例的目的是介绍dsolve函数最简单的调用格式。其在MATLAB 2009a版本下的求解代码如下：

```
S = dsolve('Dx = y,Dy = - x')
S =
    y: [1x1 sym]
    x: [1x1 sym]
>> S.x
ans =
(C2 * i)/exp(i * t) - C1 * i * exp(i * t)
>> S.y
ans =
C1 * exp(i * t) + C2/exp(i * t)
```

如果在2008a版本下，相应的结果如下：

```
S = dsolve('Dx = y,Dy = - x')
S =
    x: [1x1 sym]
    y: [1x1 sym]
>> S.x
ans =
- C1 * cos(t) + C2 * sin(t)
>> S.y
ans =
C1 * sin(t) + C2 * cos(t)
```

上述结果中的 $C1$ 和 $C2$ 都是任意常数。从以上结果可以看出，在2009a和2008a版本下求得的结果形式还是有一些差异的。虽然求解结果一样，但是形式的差异体现了Mupad和Maple两个不同符号计算引擎的差异。

【例10.1-2】 画图展示微分方程 $y=xy'-(y')^2$ 的通解和奇异解的关系。通过本例，读者将看到：R2008a和R2009a下dsolve求解的异同；未指定独立变量时的求解结果；如何将符号结果转化成数值表达式；一些绘图指令的应用。

以下代码基于 MATLAB R2009a 版本。

```
>> y = dsolve('y = x * Dy - (Dy)^2','x') % 注意书写规则,本例需要指定独立变量为 x
y =
        x^2/4
C3 * x - C3^2
% 利用 str2func 函数转化符号表达式为匿名函数,vectorzie 函数将其转化成"元素运算"形式,
% 即通常的"点运算"形式
>> f1 = str2func(['@(x)' vectorize(char(y(1)))])
f1 =
    @(x)x.^2./4
>> x1 = -6:0.2:6;
>> y1 = f1(x1);
% 画奇异解,LineH 为奇异解曲线的句柄,读者可以运行 set(lineH)观察其可设置的属性,
% set(LineH,'属性名',属性值)来设置某属性
>> lineH = plot(x1,y1,'color','r','LineWidth',5,'LineStyle','-.');
>> f2 = str2func(['@(C3) @(x)' vectorize(char(y(2)))]) % 构造双重匿名函数
f2 =
    @(C3)@(x)C3. * x - C3.^2
>> hold on
>> for C3 = -2:0.5:2
f2C3 = f2(C3); % 对于每个具体的 C3,得到相应的通解的具体解
plot(x1,f2C3(x1));
end
>> hold off
>> title('\fontname{隶书}\fontsize{16}微分方程通解和奇异解')
```

得到的图形如图 10.1 所示。

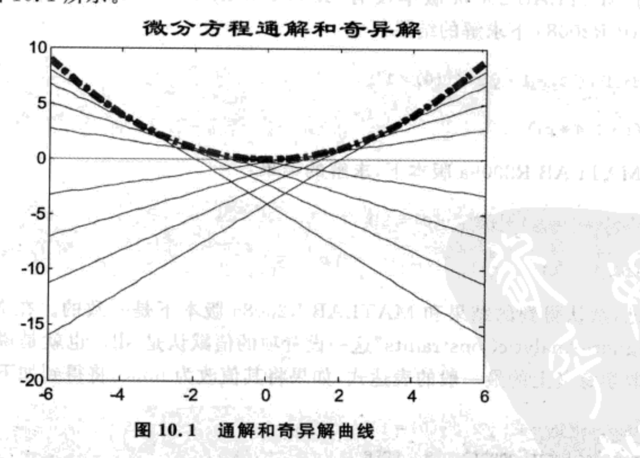


图 10.1 通解和奇异解曲线

注意 如果不指定独立变量为 x,MATLAB 还会认为默认变量为 t,这样得出的解如下:

```
y = dsolve('y = x * Dy - (Dy)^2')

y =

                                   0
x^2/4 - (x + exp((2 * C7 + t - x)/x)/exp(wrightOmega((2 * C7 + t - x + x * log(1/x))/x
)))^2/4
x^2/4 - (x - exp((2 * C3 + t - x)/x)/exp(wrightOmega((2 * C3 + t - x + x * log( - 1/x))/x
)))^2/4
```

上述代码是基于 MATLAB 2009a 版本的，如果在 MATLAB 2008a 版本下，直接运行：

```
y = dsolve('y = x * Dy - (Dy)^2','x')
```

会报错，这时候需要写成"导数在前函数在后，导数阶数降阶"的形式，即下述表达式：

```
y = dsolve('(Dy)^2 - x * Dy + y = 0','x')
```

运行结果如下：

```
y = dsolve('(Dy)^2 - x * Dy + y = 0','x')
y =
    1/4 * x^2
 - C1^2 + x * C1
```

剩下的画图等操作可以参考上述 MATLAB 2009a 版本下的代码。

【例 10.1－3】 求解微分方程 $\frac{dy}{dt}=1+y^2$，$y(0)=1$。本例展示"IgnoreAnalyticConstraints"设置项的用法。

由于 MATLAB 2008a 版本没有"IgnoreAnalyticConstraints"这一设置项，先来看看在 MATLAB R2008a 下求解的结果：

```
y = dsolve('Dy = 1 + y^2','y(0) = 1')
y =
tan(t + 1/4 * pi)
```

在 MATLAB R2009a 版本下，求解的结果：

```
y = dsolve('Dy = 1 + y^2','y(0) = 1')
y =
tan(pi/4 + t)
```

可见，默认得到的结果和 MATLAB R2008a 版本下是一致的。在 MATLAB R2009a 版本下，"IgnoreAnalyticConstraints"这一设置项的值默认是 all。也就是说，得到的表达式并不是严格数学意义上的最一般的表达式，如果将其值改为 none，将得到如下结果：

```
y = dsolve('Dy = 1 + y^2','y(0) = 1',...
'IgnoreAnalyticConstraints','none')
y =
piecewise([C20 in Z_,tan(pi/4 + t + pi * C20)])
```

上述结果是数学意义上严格条件下微分方程的通解。

【例 10.1-4】　求解两点边值问题：$xy''-3y'=x^2$，$y(1)=0$，$y(5)=0$。前面都是微分方程初值问题的例子，本例介绍 dsolve 求解边值问题。继续介绍符号表达式如何可视化。

以下代码基于 MATLAB R2009a 版本：

```
>> y = dsolve('x * D2y - 3 * Dy = x^2','y(1) = 0,y(5) = 0','x')
y =
(31 * x^4)/468 - x^3/3 + 125/468
% 用另一种方法将符号表达式转化成匿名函数，读者可以对照例 10.1-2 中的方法
>> eval(['f = @(x) ',vectorize(char(y))])
f =
    @(x)(31. * x.^4)./468 - x.^3./3 + 125./468
>> x = -1:0.2:6;
>> LineH = plot(x,f(x)); % 画微分方程解曲线，返回曲线句柄 LineH
% 通过曲线的句柄设置曲线的一些属性值
>> set(LineH,'color',[0 1 1],'LineWidth',3,'LineStyle','--')
>> hold on
>> plot([1 5],[0,0],'*','color','r','markersize',12) % 画微分方程的两个边值点
>> text(1,1,'y(1) = 0') % 图上标注边值条件
>> text(4,1,'y(5) = 0')
>> title(['x * D2y - 3 * Dy = x^2',', y(1) = 0,y(5) = 0'])
>> hold off
```

运行上述代码得到的图形如图 10.2 所示。

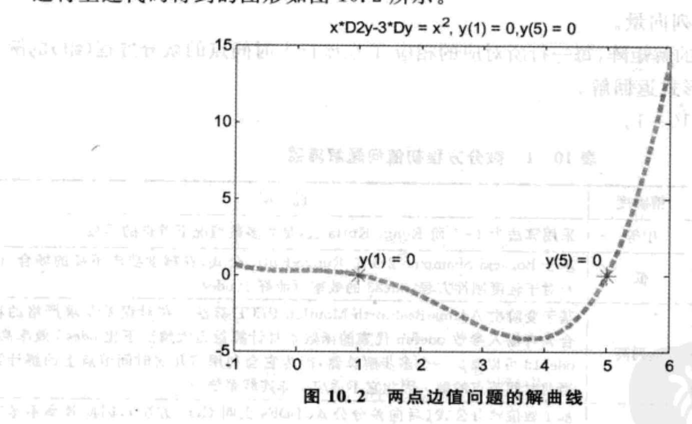


图 10.2　两点边值问题的解曲线

10.2　数值求解常微分方程函数

10.2.1　概　述

在 10.1 节，介绍了符号求解各类型的微分方程(组)，但是能够求得解析解的微分方程往往只是出现在大学课堂上，在实际应用中，绝大多数微分方程(组)无法求得解析解。这就需要利用数值方法求解。MATLAB 以数值计算见长，提供了一系列数值求解微分方程的函数。

这些函数可以求解非刚性问题、刚性问题、隐式微分方程、微分代数方程等初值问题，也可以求解延迟微分方程以及边值问题等。本节对这些函数做一些简单介绍，10.3 节开始以案例形式介绍上述各类型微分方程的求解方法。

10.2.2 初值问题求解函数

MATLAB R2009a 提供了以下函数用于各种初值问题求解：ode23，ode45，ode113，ode15s，ode23s，ode23t，ode23tb。这些函数的统一调用格式如下：

```
[T,Y] = solver(odefun,tspan,y0)
[T,Y] = solver(odefun,tspan,y0,options)
sol = solver(odefun,[t0 tf],y0...)
```

输入参数说明：

odefun：表示微分方程(组)的句柄，本书后面章节将会给出使用范例。

tspan：微分方程(组)的求解时间区间，有两种格式[t0,tf]或者[t0,t1,…,tf]，两者都以 t0 为初值点，根据 tf 自动选择积分步长。前者返回实际求解过程中所有求解的时间点上的解，而后者只返回设定的时间点上的解。后者对计算效率没有太大影响，但是求解大型问题时，可以减少内存存储。

y0：是微分方程(组)的初值，即所有状态变量在 t0 时刻的值。

options：结构体，通过 odeset 设置得到的微分优化参数，是一个结构体数组。

返回参数说明：

T：时间点组成的列向量。

Y：微分方程(组)的解矩阵，每一行所对应的相应 T 在该行上时间点的微分方程(组)的解。

sol：以结构体的形式返回解。

各函数介绍见表 10－1。

表 10－1 微分方程初值问题解算器

函 数	问题类型	精确度	说 明
ode45	非刚性	中等	采用算法为 4－5 阶 Runge-Kutta 法，是大多数情况下首选的函数
ode23	非刚性	低	基于 Bogacki-Shampine 2-3 阶 Runge-Kutta 公式，在精度要求不高的场合，以及对于轻度刚性方程，ode23 的效率可能好于 ode45
ode113	非刚性	低到高	基于变阶次 Adams-Bashforth-Moutlon PECE 算法。在对误差要求严格的场合或者输入参数 odefun 代表的函数本身计算量很大情况下比 ode45 效率高。ode113 可以看成一个多步解算器，因为它会利用前几次时间节点上的解计算当前时间节点的解。因此它不适应于非连续系统
ode15s	刚性	低到中	基于数值差分公式(后向差分公式，BDFs 也叫 Gear 方法)，因此效率不是很高。同 ode113 一样，ode15s 也是一个多步计算器。当 ode45 求解失败，或者非常慢，并且怀疑问题是刚性的，或者求解微分代数问题时可以考虑用 ode15s
ode23s	刚性	低	基于修正的二阶 Rosenbrock 公式。由于是单步解算器，当精度要求不高时，它的效率可能会高于 ode15s。ode 23s 可以解决一些 ode15s 求解起来效率不太高的刚性问题
ode23t	适度刚性	低	ode23t 可以用来求解微分代数方程
ode23tb	刚性	低	当方程是刚性的，并且求解要求精度不高时可以使用

关于各函数优化选项 option 的参数设置,读者可以参考帮助文档,这里限于篇幅不再详述。

10.2.3 延迟问题以及边值问题求解函数

对于延迟问题的求解,MATLAB 提供了 dde23 和 ddesd 函数。前者用来求解状态变量延迟为常数的微分方程(组),后者用来求解状态变量延迟不为常数的微分方程(组)。调用格式以及参数意义大部分类似于 ode 系列求解函数,不同的是要输入延迟参数以及系统在时间小于初值时的状态函数,详细格式可以参考帮助文档。

对于边值问题的求解,MATLAB 提供了两个求解函数 bvp4c 和 bvp5c,后者求解精度要比前者好。如果以 bvpsolver 表示 bvp4c 或者 bvp5c,那么这两个函数有着统一的调用格式:

```
solinit = bvpinit(x, yinit, params)
sol = bvpsolver(odefun,bcfun,solinit)
sol = bvpsolver(odefun,bcfun,solinit,options)
```

输入输出各参数说明:

solinit:包含解的初始猜测值的结构数组,由 bvpinit 生成。

x:需要计算的按顺序排列的初始节点。类似于初值问题 ode ** 系列函数输入参数中的 tspan。对于[a,b]上的边值问题,x(1)=a,x(end)=b。对于多点边值问题,如[a,c,b],a、b、c 上都有边值限制,那么 x=[a,…,c,c,…,b],即中间的边值点要重复。在 a 和 c 之间,c 和 b 之间其他的点可有可无。

yinit:解的初始猜测值,可以是具体值,也可以是函数,类似 ode ** 的 x0。

params:其他的未知参数,也是一个猜测值。

odefun:描述边值问题微分方程的函数句柄,类似 ode ** 中的 odefun。

bcfun:边值函数句柄,一般是两点边值形式,也支持多点边值形式(具体可以参考帮助文档)。

options:边值问题解算器的优化参数,可以通过 bvpset 设置(具体参数可以查看帮助文档)。

本章后半部分将针对几种类型的微分方程以案例形式给出其求解示例。

10.2.4 求解前的准备工作

很多 MATLAB 初学者都爱用符号法求解微分方程而不愿意用数值方法求解微分方程。其主要原因就在于,数值求解的时候要把求解的微分方程转化成 MATLAB 数值求解函数可以认识的形式,这种转化比符号求解要麻烦一些。

但是在理解并明白转化的含义后,掌握转换的方法并不难,而且各微分方程数值求解函数所接受的输入形式是一致的。

常微分方程的形式是多种多样的。一般来说,很多高阶微分方程可以通过变量替换转化成一阶微分方程组,即可以写成下面的形式:

$$\boldsymbol{M}(t,y)y'=\boldsymbol{F}(t,y) \tag{10-1}$$

$\boldsymbol{M}(t,y)$被称为质量矩阵，如果其为非奇异矩阵，则式(10-1)可以写成

$$y'=\boldsymbol{M}^{-1}(t,y)\boldsymbol{F}(t,y) \tag{10-2}$$

将等式(10-2)右半部分用 odefun 表示出来(具体表现形式可以采用匿名函数、子函数、嵌套函数、单独 M 文件等形式)，就是 ode45，ode23 等常微分方程初值问题求解的输入参数 odefun。

如果$\boldsymbol{M}(t,y)$为奇异矩阵，则式(10-1)称为微分代数方程组(differential algebraic equations, DAEs)，可以利用求解刚性微分方程的函数(如 ode15s，ode23s 等)来求解。从输入形式上看，求解 DAEs 和求解普通的 ODE 很类似，主要区别是需要给微分方程求解器指定质量矩阵$\boldsymbol{M}(t,y)$。关于其原理，有兴趣的读者可以参考文献[23]，这里不再详述。本章后面部分将给出 DAEs 的求解案例。

还有一些方程无法写成式(10-1)或者式(10-2)的形式，即隐式微分方程，关于其解法也将在后面进行介绍。

10.3 案例20：非刚性/刚性常微分方程初值问题求解

10.3.1 概 述

本案例着重介绍非刚性(nonstiff)/刚性(stiff)常微分方程求解。刚性/非刚性问题最直观的判别方法就是从解在某段时间区间内的变化来看。非刚性问题变化相对缓慢，而刚性问题在某段时间内会发生剧烈变化，即在很短的时间内，解的变化巨大。对于刚性问题不适合用 ode45 来求解，如果非要用 ode45 来求解，则达到指定精度所耗费的时间往往会非常长。

10.3.2 非刚性问题举例

【例 10.3-1】 求如下微分方程的解，求解时间区间为[0,30]，并图示。

$$\frac{d^2x}{dt^2}-\mu(1-x^2)\frac{dx}{dt}+x=0$$

$$x(0)=1,\frac{dx(0)}{dt}=0$$

其中，μ未知，是计算过程中生成的数值。

将所给方程转化成式(10-2)的形式，进行变量替换，令$y_1=x, y_2=\frac{dx}{dt}$。于是，原方程可以改写成如下一阶微分方程组形式：

$$\begin{bmatrix}\frac{dy_1}{dt}\\ \frac{dy_2}{dt}\end{bmatrix}=\begin{bmatrix}y_2\\ \mu(1-y_1^2)y_2-y_1\end{bmatrix},\quad \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix}=\begin{bmatrix}1\\ 0\end{bmatrix}$$

由于μ未知，因此在构造微分函数的句柄 odefun 时需要考虑μ，即能根据生成的μ实时得到与之对应的微分方程，将求解微分方程并画图的一系列过程写到一个 M 文件中，代码如下：

```
function example10_3_1
close all;
% 利用 nested function 形式来表示微分方程
    function ydot = DyDtNestedFun(t,y)
        ydot = [y(2);mu * (1 - y(1)^2) * y(2) - y(1)];
    end
% 利用匿名函数形式来表示微分方程
DyDtAnony = @(mu)@(t,y)[y(2);mu * (1 - y(1)^2) * y(2) - y(1)];
% 以下代码对 mu = 1,2,3 分别求解微分方程,并画出微分方程的解
tspan = [0,30]; % 时间区间
y0 = [1 0];
figure(1);h1 = axes;hold on;
figure(2);g1 = axes;hold on;
ColorOrder = get(gca,'ColorOrder'); % 默认的坐标轴曲线颜色顺序
% 曲线类型顺序,方便成书时区分显示不同线
LineStyle = {'-','--',':'};
for mu = 1:3
    % 利用 ode45 求解微分方程(输入微分方程组是 nested function 形式表示的)
    [tt yy] = ode45(@DyDtNestedFun,tspan,y0);
    % 在句柄值为 h1 的坐标轴上根据不同的 mu 画相应的解曲线
    plot(h1,tt,yy(:,1),'color',ColorOrder(mu,:),'LineStyle',LineStyle{mu});
    % 在句柄值为 g1 的坐标轴上根据不同的 mu 画相应的平面相轨迹
    plot(g1,yy(:,1),yy(:,2),'color',ColorOrder(mu,:),'LineStyle',LineStyle{mu});
end
xlabel(h1,'t'); % 在句柄值为 h1 的坐标轴的 x 轴上标注
title(h1,'x(t)');
legend('\mu = 1','\mu = 2','\mu = 3');hold off
xlabel(g1,'位移');ylabel(g1,'速度');hold off
% 下面利用 ode45 求解微分方程(输入的微分方程组是匿名函数形式表示的)
figure(3);h2 = axes;hold on;
figure(4);g2 = axes;hold on;
for mu = 1:3
    [tt yy] = ode45(DyDtAnony(mu),tspan,y0);
    plot(h2,tt,yy(:,1),'color',ColorOrder(mu,:),'LineStyle',LineStyle{mu});
    plot(g2,yy(:,1),yy(:,2),'color',ColorOrder(mu,:),'LineStyle',LineStyle{mu});
end
xlabel(h2,'t');
title(h2,'x(t)');
legend('\mu = 1','\mu = 2','\mu = 3');hold off
xlabel(g2,'位移');ylabel(g2,'速度');hold off
end
```

运行上述代码会发现,无论采用 nest function 表示微分方程组还是用匿名函数表示微分方程组,都得到了一模一样的如图 10.3 所示的微分方程解曲线和如图 10.4 所示的平面相轨迹。采用 nested function 表示微分方程可以方便地与主程序之间共享 mu,而且 nested function 在主函数之内的位置也是随意的,甚至可以放到主函数内的 mu 出现之前。主函数内的

mu 生成之后或者改变值之后调用微分方程 DyDtNestedFun，DyDtNestedFun 内的 mu 也会作出改变。而采用双重匿名函数来构造微分方程是把 mu 作为外重变量传入的，给定一个 mu，DyDtAnony(mu)就会生成对应的微分方程句柄。

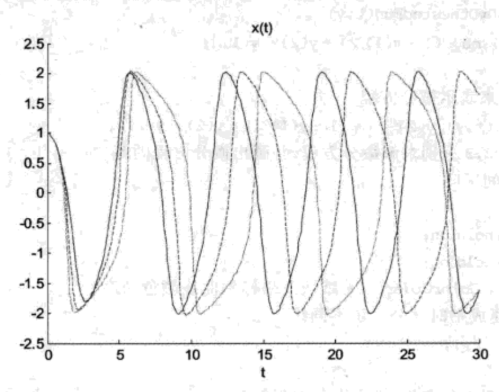


图 10.3　微分方程解

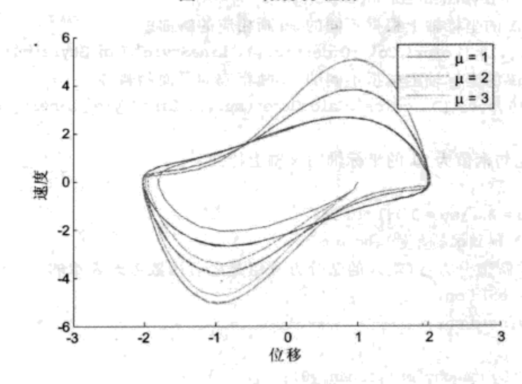


图 10.4　平面相轨迹

10.3.3　刚性问题举例

【例 10.3-2】　求解如下微分方程组：

$$\begin{bmatrix}\dfrac{dy_1}{dt}\\ \dfrac{dy_2}{dt}\end{bmatrix}=\begin{bmatrix}a-(b+1)y_1+y_1^2y_2\\ by_1-y_1^2y_2\end{bmatrix},\quad \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix}=\begin{bmatrix}3\\ 4\end{bmatrix}$$

其中，$a=100$；$b=50$。

这里也涉及参数 a 和 b 传递到微分方程的问题，上节已经给出两种传递参数的方式。这里给出参数传递的另一种途径，即在调用微分方程求解函数的时候传递参数，只不过这种情况

下，需要把输入参数都写全，待传递的参数写在后面。该问题的求解代码如下：

```
function example10_3_2
tspan = [0,10]; % 变量求解区间
y0 = [3 4]; % 初值
a = 100; % 参数赋值
b = 50;
options = odeset('RelTol',0.001); % 设置相对误差
tic;
% 需要把 ode45 输入参数都写全,待传递的参数 a,b 写在后面。下面 ode23,ode15s 类似
[t45,y45] = ode45(@DyDtSubFun,tspan,y0,options,a,b);
time45 = toc;
disp(['ode45 计算点数(子函数表示微分方程):',num2str(length(t45)),...
    ';所用时间:',num2str(time45),'s.'])
tic;
[t23,y23] = ode23(@DyDtSubFun,tspan,y0,options,a,b);
time23 = toc;
disp(['ode23 计算点数(子函数表示微分方程):',num2str(length(t23)),...
    ';所用时间:',num2str(time23),'s.'])
tic;
[t15s,y15s] = ode15s(@DyDtSubFun,tspan,y0,options,a,b);
time15s = toc;
disp(['ode15s 计算点数(子函数表示微分方程):',num2str(length(t15s)),...
    ';所用时间:',num2str(time15s),'s.'])
% 用匿名函数表示微分方程
DyDtAnony = @(t,y,a,b) [a - (b+1) * y(1) + y(1).^2 * y(2);b * y(1) - y(1).^2 * y(2)];
tic;
[t15sAnony,y15sAnony] = ode15s(DyDtAnony,tspan,y0,options,a,b);
time15sAnony = toc;
disp(['ode15s 计算点数(匿名函数表示微分方程)::',num2str(length(t15sAnony)),...
    ';所用时间:',num2str(time15sAnony),'s.'])
% 画图展示
figure;
subplot(131);
plot(t23,y23,'k-');xlabel('\itt','fontsize',16);
title('子函数形式/ode23')
subplot(132);
plot(t15s,y15s,'k--');xlabel('\itt','fontsize',16);
title('子函数形式/ode15s')
subplot(133);
plot(t15sAnony,y15sAnony,'k:');xlabel('\itt','fontsize',16);
title('匿名函数形式/ode15s')

% 用子函数来表示微分方程
function dy = DyDtSubFun(t,y,a,b)
dy(1,1) = a - (b+1) * y(1) + y(1).^2 * y(2);
dy(2,1) = b * y(1) - y(1).^2 * y(2);
```

运行上述代码得到如下结果：

```
ode45 计算点数(子函数表示微分方程):102869;所用时间:7.8812s.
ode23 计算点数(子函数表示微分方程):33968;所用时间:4.713s.
ode15s 计算点数(子函数表示微分方程):90;所用时间:0.041823s.
ode15s 计算点数(匿名函数表示微分方程):90;所用时间:0.044306s.
```

从上面的代码可以看到，ode45 以及 ode23 求解的计算量都比 ode15s 大得多，因此耗费时间也长得多，利用子函数或者匿名函数表示微分方程都可以在方程求解的时候把参数传递到方程里去。此外，感兴趣的读者可以改变参数 a 和 b 的值，会发现，随着 b/a 逐渐增大，方程的刚性会逐渐增强。图 10.5 是采用不同的微分方程求解器以及不同的微分方程表示方法得到的微分方程解随时间变化的图像。从图 10.5 中可以看到，其结果都是一致的。

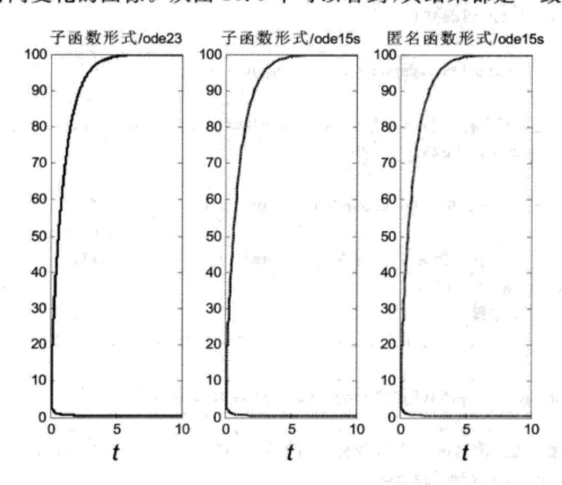


图 10.5　不同求解器得到的微分方程的解

【例 10.3－3】 已知 $y_1(t)$，$y_2(t)$满足如下方程式：

$$\begin{bmatrix}\dfrac{\mathrm{d}y_1}{\mathrm{d}t}\\ \dfrac{\mathrm{d}y_2}{\mathrm{d}t}\end{bmatrix}=\begin{bmatrix}y_2-f(t)\\ y_1g(t)-y_2\end{bmatrix},\quad \begin{bmatrix}y_1(0)\\ y_2(0)\end{bmatrix}=\begin{bmatrix}1\\ 2\end{bmatrix}$$

$$f(t)=\begin{cases}\sin(t), & t<4\pi\\ 0, & t\geqslant 4\pi\end{cases},\quad g(t)=\begin{cases}0, & t<7\pi/2\\ 2\cos(t), & t\geqslant 7\pi/2\end{cases}$$

求：使得 $F(t)=y_1(t)+y_2(t)$值为 0 的时间点。

本例目的：介绍 ode23tb 和函数 deval 的使用，以及如何寻找一个关于微分方程解的复合函数的零点。其代码如下：

```
function T0 = example10_3_3
 % 用 nested function 表示微分方程
    function dy = DyDxNestedFun(t,y)
        ft = 0;
        gt = 0;
        if t<4 * pi
            ft = 2 * sin(t);
        end
        if t>3.5 * pi
            gt = cos(t);
        end
        dy = [y(2) - ft;y(1) * gt - y(2)];
    end
tspan = [0,20]; % 变量求解区间
y0 = [1,2]; % 初值
sol = ode23tb(@DyDxNestedFun,tspan,y0); % 调用 ode23tb 求解
subplot(121);
plot(sol.x,sol.y(1,:),'k-','linewidth',2); % 画出函数 y1(t)曲线
hold on;
plot(sol.x,sol.y(2,:),'k-.','linewidth',2); % 画出函数 y2(t)曲线
hold off;
% 图例,自动选择最佳位置
L1 = legend('{\ity}_1(t)','{\ity}_2(t)','Location','best');
set(L1,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);
subplot(122);
plot(sol.x,sum(sol.y),'linewidth',2);hold on;
plot([0,20],[0 0],'color','r','linestyle',':','linewidth',2);
xlabel('\itt','fontsize',16);
% 图例,自动选择最佳位置
L2 = legend('F(t) = {\ity}_1(t)  +  {\ity}_2(t)','y = 0 直线','Location','best');
set(L2,'fontname','Times New Roman');
hold off;
% 以下代码利用 arrayfun + fzero + deval 实现求 F(t)的所有零点
T0 = arrayfun(@(t0) fzero(@(t) sum(deval(sol,t)),t0),[2 4 6 10 18]);
end
```

运行 T0＝example10_3_3 得到 $F(t)=y_1(t)+y_2(t)$ 的五个零点为：

```
T0 = example10_3_3
T0 =
2.0961   4.1875   4.1875   10.4707   16.1420
```

得到 $y_1(t)$，$y_2(t)$ 的图像以及 $F(t)=y_1(t)+y_2(t)$ 的图像如图 S 所示。

从上面的例子可以看出，对于较为复杂的微分方程，完全可以方便地用 nested function 来表示。deval 函数的作用是通过微分方程求解函数得到的结构体 sol 计算和指定的 t 对应的 x 值，和 polyval 等函数很相似。从图 10.6 可以看到 $F(t)=y_1(t)+y_2(t)$ 一共有五个零点，因此，在 example10_3_3 中最后求 $F(t)=y_1(t)+y_2(t)$ 的零点时，初值都设置在相应的零点附近。

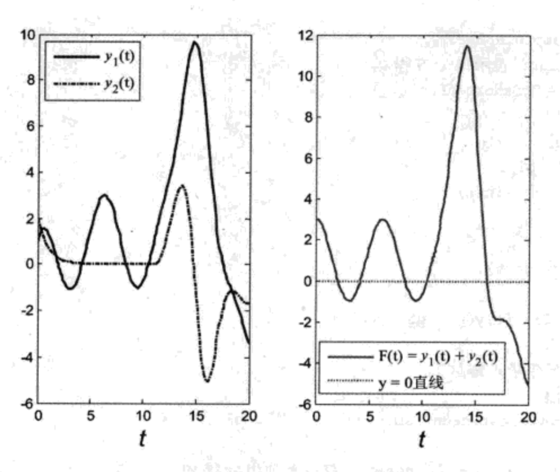


图 10.6 微分方程解曲线以及两个解的和曲线

10.4 案例21：隐式微分方程(组)求解

10.4.1 概 述

一些微分方程(组)在初始给出的时候是不容易显示地表示成上面提到的标准形式的。这时候就需要想办法表示成上述的形式。一般来说有三种思路，一种是利用 solve 函数符号求解出高阶微分的显式表达式；另一种是利用 fzero/fsolve 函数求解状态变量的微分值；还有一种是利用 MATLAB 自带的 ode15i 函数。下面逐一介绍这些方法。

10.4.2 利用 solve 函数

【例 10.4-1】 求下列微分方程组的解。

$$\begin{cases} y_2'[y_2\cos(4t)-y_1^3]-\dfrac{t}{5}y_1'=0 \\ t\sin(y_2)/8-2y_2y_2'+\sqrt{t}y_1'=0 \end{cases}$$

初始条件为：$y_1(1)=1, y_2(1)=1$。

该方程组的 y_1' 和 y_2' 并不是显式给出的。当然，针对本例，可以通过手算得出 y_1' 和 y_2' 的表达式。不过，为了一般起见，更是为了介绍 solve 函数符号求解的过程，本例采用 solve 函数求解 y'_1 和 y'_2 的显示表达式，并自动将符号表达式转化成微分方程函数句柄，供 ode45 等函数调用求解。其程序如下：

```
function example10_4_1
% 用符号积分求解 dy 的解析表达式
[dy(1,1),dy(2,1)] = solve('dy2 * (y(2) * cos(4 * t) - y(1)^3) - t/5 * dy1',...
    't * sin(y(2))/8 - 2 * y(2) * dy2 + sqrt(t) * dy1','dy1','dy2');
% 利用字符串执行函数 eval 生成微分方程的匿名函数
```

```
eval(['DyDtAnony = @(t,y) ['char(dy(1,1)),';',char(dy(2,1)),']']);
tspan = [1 30];
y0 = [1;1];
[t,y] = ode45(DyDtAnony,tspan,y0); % 调用 ode45 求解
figure;
plot(t,y(:,1),'k-');
hold on
plot(t,y(:,2),'k:');
% 图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);
```

上述程序得到的图形如图 10.7 所示。

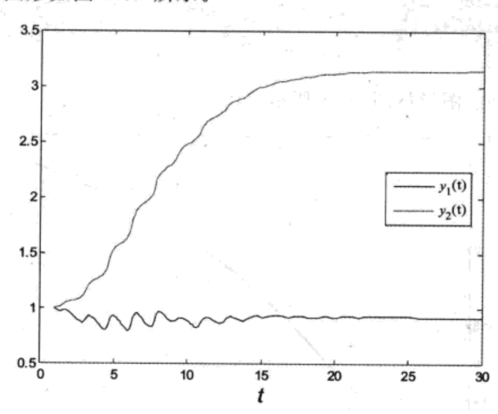


图 10.7　函数 ode45 求解的结果曲线

10.4.3　利用 fzero/fsolve 函数

不是所有的隐式微分方程(组)都能像例 10.4-1 那样可以用 solve 函数解出显式表达式，而这样的微分方程(组)就是真正的隐微分方程(组)。

这时候该怎么办呢？首先分析一下微分方程解算器求解显式微分方程的做法。微分方程解算器对微分方程的识别是通过 odefun 来完成的，构造好的 odefun 的输出就是状态变量的一阶微分，只不过显式条件下，这个值可以直接通过算式表达出来。微分方程解算器实际上是不管 odefun 内部是如何得到输出的。因此，如果微分方程没有显式表达式，可以用数值求解状态变量的一阶微分。对于含有一个状态变量的隐式微分方程可以通过 fzero 函数求解，而隐式微分方程组可以通过 fsolve 函数求解。

【例 10.4-2】　求下列微分方程的解。

$$y'=\exp\left\{-\left\{[y-0.5-\exp(-t+y')]^2+y^2-\frac{t}{5}+3\right\}\right\}$$

其中，初始条件为 $y(0)=0.1$，求解时间区间为[0,20]。

该微分方程是真正的隐式微分方程，没有y'关于t、y的显式表达式，只能通过数值方法求出y'关于t、y的表达式。其代码如下：

```
function example10_4_2
% nested function 构造微分方程，利用 fzero 求解隐式微分方程输出项
    function DyDt = DyDtNestedFun(t,y)
        fun = @(yp) yp - exp( - (( y - 0.5 - exp( - t + yp))^2 + y^2 - t/5 + 3));
        DyDt = fzero(fun,3);
    end
tspan = [0,20]; % 时间区间
y0 = 0.1;
[t y] = ode45(@DyDtNestedFun,tspan,y0);
figure;
plot(t,y,'k-','linewidth',2);
xlabel('\itt','fontsize',16);
end
```

运行上述程序得到的图形如图 10.8 所示。

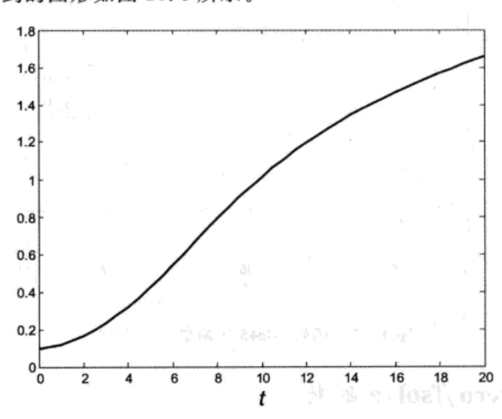


图 10.8　隐式微分方程求解示例图像

【例 10.4-3】 求下列微分方程组的解。

$$\begin{cases} x''\sin(y')+y''=-2xy+xx''y' \\ xx''y''+\cos(y'')=3x'y \end{cases}$$

初始条件为

$$x(0)=1, x'(0)=0, y(0)=0, y'(0)=1$$

首先令$y_1(t)=x(t), y_2(t)=x'(t), y_3(t)=y(t), y_4(t)=y'(t)$，则上述方程组变为

$$\begin{cases} y_1'=y_2 \\ y_2'\sin(y_4)+(y_4')^2=-2y_1y_3+y_1y_2'y_4 \\ y_3'=y_4 \\ y_1y_2'y'_4+\cos(y_4')=3y_2y_3 \end{cases}$$

上面是一个隐式微分方程组，在构造微分方程组函数句柄时，需要对y_2'和y_4'利用 fsolve

函数进行数值求解。接下来再用 ode45 等函数求解微分方程组。相应的代码如下：

```
function example10_4_3
% nested function 构造微分方程组,利用 fsolve 求解隐式微分方程组的某些输出项
    function DyDt = DyDtNestedFun(t,y)
        fun = @(dy24)[dy24(1) * sin(y(4)) + dy24(2)^2 + 2 * y(1) * y(3) - y(1) * dy24(1) * y(4);
            y(1) * dy24(1) * dy24(2) + cos(dy24(2)) - 3 * y(2) * y(3)];
        options = optimset('display','off');
        % 使用 fsolve 求解出与原问题对应的 x"和 y"
        dy24Zero = fsolve(fun,y([1,3]),options);
        DyDt = [y(2);dy24Zero(1);y(4);dy24Zero(2)]; % 状态变量一阶微分值
    end
tspan = [0,5]; % 时间区间
y0 = [1 0 0 1]';
[t y] = ode45(@DyDtNestedFun,tspan,y0);
% 画图展示结果
figure;
plot(t,y(:,1),'k-','linewidth',2);
hold on
plot(t,y(:,2),'k--','linewidth',2);
plot(t,y(:,3),'k-.','linewidth',2);
plot(t,y(:,4),'k:','linewidth',2);
% 图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)'...
    ,'{\ity}_4(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);
end
```

运行下述代码,得到程序执行时间和如图 10.9 所示的图形。

```
>> tic;example10_4_3;toc
Elapsed time is 0.986295 seconds.
```

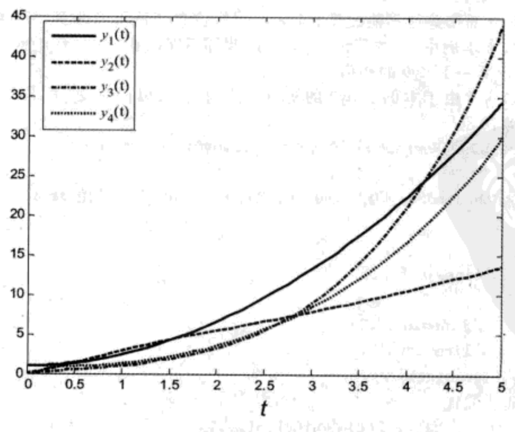


图 10.9　隐式微分方程组求解示例图像

在 ode15i 出现之前，对于真正的隐式微分方程，一般都用上述办法来进行求解。这种方法的主要缺点是每一个计算节点上的处于隐变量位置的微分值都要进行数值求解，当系统比较庞大时，计算量非常大。而且，由于初始值的选取问题，fsolve 函数并不是每次都可以成功求解。

10.4.4 利用 ode15i 函数

从 MATLAB 7.0 版本开始，MATLAB 提供了 ode15i 函数用来求解隐式微分方程乃至微分代数方程(DAE)。下面介绍 ode15i 函数的用法。

【例 10.4－4】 用 ode15i 函数求下列微分方程组的解。

$$\begin{cases} y_1' = y_2 \\ y_2'\sin(y_4) + (y_4')^2 = -2y_1y_3 + y_1y_2'y_4 \\ y_3' = y_4 \\ y_1y_2'y_4' + \cos(y_4') = 3y_2y_3 \end{cases}$$

初始条件为：

$$y_1(0)=1,\quad y_2(0)=0,\quad y_3(0)=0,\quad y_4(0)=1$$

该例是例 10.4－3 进行变量替换后的形式。下面利用 ode15i 函数对其进行求解。其代码如下：

```
function example10_4_4
% nested function 构造微分方程组，注意多了一个一阶导数变量 dy(与非隐式微分方程不同)
    function DyDt = DyDtNestedFun(t,y,dy)
        DyDt = [dy(1) - y(2);
            dy(2) * sin(y(4)) + dy(4)^2 + 2 * y(1) * y(3) - y(1) * dy(2) * y(4)
            dy(3) - y(4)
            y(1) * dy(2) * dy(4) + cos(dy(4)) - 3 * y(3) * y(2)];
    end
t0 = 0; % 自变量的初值
y0 = [1;0;0;1]; % 初值 y0
% fix_y0 表明初值 y0 的哪些值不能改变。1 表示对应位置初值不能改变,0 为可以改变
fix_y0 = ones(4,1); % 本例中 y0 的值都给出了,因此都不能改变,所有 fix_y0 全为 1
dy0 = [0 3 1 0]'; % 猜测一下 dy0 的初值;
fix_dy0 = zeros(4,1); % 由于本例中 dy0 的初值是猜测的,可以都改变,因此 fix_dy0 全部为 0
% 调用 decic 函数来决定
[y02,dy02] = decic(@DyDtNestedFun,t0,y0,fix_y0,dy0,fix_dy0);
% 求解微分方程
[t y] = ode15i(@DyDtNestedFun,[0 5],y02,dy02); % y02 和 dy02 是由 decic 输出参数
% 画图展示结果
figure;
plot(t,y(:,1),'k-','linewidth',2);
hold on
plot(t,y(:,2),'k--','linewidth',2);
plot(t,y(:,3),'k-.','linewidth',2);
plot(t,y(:,4),'k:','linewidth',2);
% 图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)',...
```

```
    ,'{\ity}_4(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);
end
```

运行下述代码，得到程序执行时间和如图 10.10 所示的图形。

```
>> tic;example10_4_4;toc
Elapsed time is 0.221251 seconds.
```

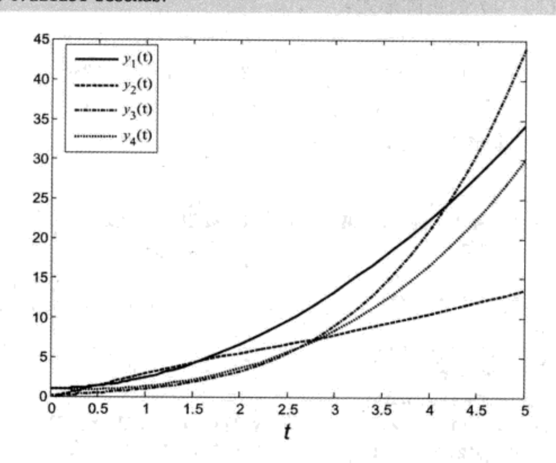


图 10.10　ode15i 函数求解隐式微分方程组示例图像

从 example10_4_3 和 example10_4_4 的执行时间来看，采用 ode15i 方法的速度明显高于用 fsolve 方法的速度，而且从图像上看，用 ode15i 方法和前面用 fsolve 方法得到的图像一致。有兴趣的读者可以比较两者得到的 y 的具体差异。需要说明的是，在给出 dy0 的初始值时，往往要猜测，如果初始猜测的值不好，也容易导致求解不成功。这种情况下可以考虑换一下猜测的初值。

10.5　案例 22：微分代数方程与延迟微分方程求解

10.5.1　概　述

在 10.2 节已经介绍了微分代数方程(DAE)以及延迟微分方程的(DDE)的一些求解函数，本案例主要结合几个方程详细介绍这些函数的用法。

10.5.2　微分代数方程举例

DAE 的求解一般有三种方法：一种是变量替换法；另一种是用 ode15s 函数；还有一种是用 10.4 节中提到的 ode15i 函数。下面结合例子对其进行介绍。

【例 10.5-1】　求解下面的微分代数方程。

$$\begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \\ y_1+y_2+y_3 \end{bmatrix} = \begin{bmatrix} -0.2y_1+y_2y_3+0.3y_1y_2 \\ 2y_1y_2-5y_2y_3-2y_2^2 \\ 1 \end{bmatrix}$$

初值条件为

$$y_1(0)=0.8,\quad y_2(0)=0.1,\quad y_3(0)=0.1$$

将上式写成类似式(10-1)的矩阵形式：

$$\boldsymbol{M}(t,y)y' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} dy_1/dt \\ dy_2/dt \\ dy_3/dt \end{bmatrix} = \begin{bmatrix} -0.2y_1+y_2y_3+0.3y_1y_2 \\ 2y_1y_2-5y_2y_3-2y_2^2 \\ y_1+y_2+y_3-1 \end{bmatrix} = \boldsymbol{F}(t,y)$$

下面给出三种方法求解上述方程的代码：

```
function example10_5_1
y01 = [0.8;0.1]; % 第一种方法的初值(第一种方法只需要用到两个变量)
y0 = [0.8;0.1;0.1]; % 初值
tspan = [0 20];
%================================
% 方法 1:变量替换
%================================
% 将 y(3)用 1 - y(1) - y(2)代替
    function DyDt = DyDtNestedFun1(t,y)
        DyDt = [ - 0.2 * y(1) + y(2) * (1 - y(1) - y(2)) + 0.3 * y(1) * y(2);
            2 * y(1) * y(2) - 5 * y(2) * (1 - y(1) - y(2)) - 2 * y(2)^2];
    end
[T1,Y1] = ode45(@DyDtNestedFun1,tspan,y01);
%==================================
% 方法 2:设置质量矩阵,用 ode15s 函数
%==================================
M = [1 0 0;0 1 0;0 0 0]; % 质量矩阵
options = odeset('mass',M);
% 被 ode15s 调用的微分函数表达式
    function DyDt = DyDtNestedFun2(t,y)
        DyDt = [ - 0.2 * y(1) + y(2) * y(3) + 0.3 * y(1) * y(2);
            2 * y(1) * y(2) - 5 * y(2) * y(3) - 2 * y(2)^2;
            y(1) + y(2) + y(3) - 1];
    end
[T2,Y2] = ode15s(@DyDtNestedFun2,tspan,y0,options);
%================================
% 方法 3:用 ode15i 函数
%================================
% 被 ode15i 调用的微分函数表达式
    function DyDt = DyDtNestedFun3(t,y,dy)
        DyDt = [dy(1) + 0.2 * y(1) - y(2) * y(3) - 0.3 * y(1) * y(2);
            dy(2) - 2 * y(1) * y(2) + 5 * y(2) * y(3) + 2 * y(2)^2;
            y(1) + y(2) + y(3) - 1];
    end
```

```
% y(1) + y(2) + y(3) - 1 = 0 表明 y0 中任何一个改变后都会至少引起其余一个发生变化,因此
y0_fix = [0;0;1]; % 任意两位都可以改为 0,比如[0;1;0]或者[1;0;0]
% 状态变量一阶微分初值,例子中没有提供,因此可以随意猜测一组值
dy0 = [1;1;1];
% 该组初值都可以改变,故全部为 0
dy0_fix = [0;0;0];
% 时间变量的初值
t0 = 0;
% 计算输入到 ode15i 解算器的 dy 以及 dy3
[y00,dy00] = decic(@DyDtNestedFun3,0,y0,y0_fix,dy0,dy0_fix);
[T3,Y3] = ode15i(@DyDtNestedFun3,tspan,y00,dy00);
%=========================
% 画图呈现三种方法的计算结果
%=========================
figure;
% 方法 1 得到的图
subplot(131);
plot(T1,Y1(:,1),'k-','linewidth',2);
hold on
plot(T1,Y1(:,2),'k-.','linewidth',2);
plot(T1,1 - Y1(:,1) - Y1(:,2),'k:','linewidth',2);
hold off
% 图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方法 1 计算结果图')
% 方法 2 得到的图
subplot(132);
plot(T2,Y2(:,1),'k-','linewidth',2);
hold on
plot(T2,Y2(:,2),'k-.','linewidth',2);
plot(T2,Y2(:,3),'k:','linewidth',2);
hold off
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方法 2 计算结果图')
% 方法 3 得到的图
subplot(133);
plot(T3,Y3(:,1),'k-','linewidth',2);
hold on
plot(T3,Y3(:,2),'k-.','linewidth',2);
plot(T3,Y3(:,3),'k:','linewidth',2);
hold off
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方法 3 计算结果图')
end
```

运行上述代码，得到如图10.11所示图形。由此可见，三种方法得到的图所示图形一致。

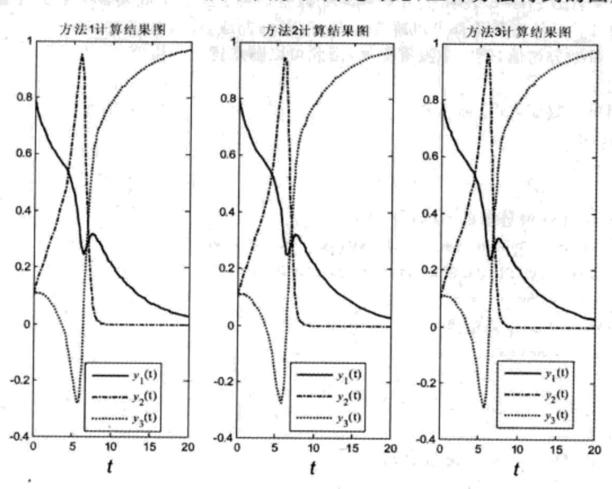


图10.11 三种方法求解DAE示例

上面是普通的微分代数方程，再看一个隐式微分代数方程的例子。

【例10.5-2】 求解下面的微分代数方程。

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_1+y_2-y_3-y_4 \end{bmatrix} = \begin{bmatrix} -0.3y_1-2y_2\sin(y_3')-y_2y_4 \\ -y_2-0.5\cos(y_1'+y_3)-0.2\sin(0.6t) \\ -0.2y_1y_2+\exp(-y_1') \\ 1 \end{bmatrix}$$

初值条件为

$$y_1(0)=1,\quad y_2(0)=0.5,\quad y_3(0)=0.3,\quad y_4(0)=0.2$$

上式是一个隐式微分代数方程，无法写成式(10-1)所示的矩阵形式，但是结合以前的案例，我们有两种方法可以求解它：一种是变量替换后用fsolve函数求解出每一计算节点的y_1'，y_2'，y_3'值，然后再调用ode45、ode23tb等函数求解；另一种方法就是直接利用ode15i函数求解。两种方法的求解代码如下：

```
function example10_5_2
y01 = [1;0.5;0.3]; % 第一种方法的初值(第一种方法只需要用到三个变量)
y0 = [1;0.5;0.3;0.2]; % 初值
tspan = [0 5];
%================================
% 方法 1:变量替换后用 fsolve 函数
%================================
% 将 y(4)用 y(1) + y(2) - y(3) - 1 代替
    function DyDt = DyDtNestedFun1(t,y)
        fun = @(dy)[dy(1) + 0.3 * y(1) + 2 * y(2) * sin(dy(3)) + y(2) * (y(1) + y(2) - y(3) - 1);
            dy(2) + y(2) + 0.5 * cos(dy(1) + y(3)) + 0.2 * sin(0.6 * t);
            dy(3) + 0.2 * y(1) * y(2) - exp( - dy(1))];
```

```
        options = optimset('display','off','TolX',1e-8,'TolFun',1e-8);
        %使用 fsolve 求解出与原问题对应的 y1',y2',y3'
        DyDt = fsolve(fun,y,options); %状态变量一阶微分值
        end
[T1,Y1] = ode45(@DyDtNestedFun1,tspan,y01);
%===============================
%方法 2:用 ode15i 函数
%===============================
%被 ode15i 调用的微分函数表达式
    function DyDt = DyDtNestedFun2(t,y,dy)
        DyDt = [dy(1) + 0.3 * y(1) + 2 * y(2) * sin(dy(3)) + y(2) * y(4);
            dy(2) + y(2) + 0.5 * cos(dy(1) + y(3)) + 0.2 * sin(0.6 * t);
            dy(3) + 0.2 * y(1) * y(2) - exp( - dy(1));
            y(1) + y(2) - y(3) - y(4) - 1];
    end
%y(1) + y(2) - y(3) - y(4) - 1 = 0 表明 y 中任何一个改变后都会至少引起其余一个发生变化,因此
y0_fix = [0;1;0;0]; %任意三位都可以改为 0,比如[0;0;1;0]或者[1;0;0;0]等
%状态变量一阶微分初值,例子中没有提供,因此可以猜测一组值,可能需要猜测好几回才能保证
%不提示"Convergence failure in DECIC."错误
dy0 = [-1; -1;2.5;0.5];
%该组初值都可以改变,故全部为 0
dy0_fix = [0;0;0;0];
%时间变量的初值
t0 = 0;
%计算输入到 ode15i 解算器的 dy 以及 dy3
[y00,dy00] = decic(@DyDtNestedFun2,0,y0,y0_fix,dy0,dy0_fix);
[T2,Y2] = ode15i(@DyDtNestedFun2,tspan,y00,dy00);
%=========================
%画图呈现三种方法的计算结果
%=========================
figure;
%方法 1 得到的图
subplot(121);
plot(T1,Y1(:,1),'k-','linewidth',2);
hold on
plot(T1,Y1(:,2),'k-.','linewidth',2);
plot(T1,Y1(:,3),'k-*','linewidth',1);
plot(T1,Y1(:,1) + Y1(:,2) - Y1(:,3) - 1,'k-o','linewidth',1);
hold off
%图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)',...
    '{\ity}_4(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方法 1 计算结果图')
%方法 2 得到的图
subplot(122);
plot(T2,Y2(:,1),'k-','linewidth',2);
hold on
plot(T2,Y2(:,2),'k-.','linewidth',2);
plot(T2,Y2(:,3),'k-*','linewidth',1);
```

```
plot(T2,Y2(:,4),'k-o','linewidth',1);
hold off
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)',...
    '{\ity}_4(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方法 2 计算结果图')
end
```

运行上述代码得到的图形如图 10.12 所示。

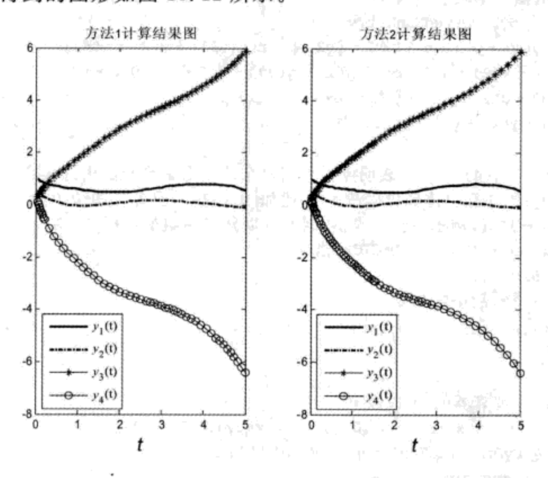


图 10.12　两种方法求解隐式 DAE 示例

【补充说明】 对于隐式微分代数方程来讲，example10_5_2 给出的两种方法各有利弊。用 fsolve 方法的最大缺点在于，某些节点上对于给定的初始值可能无法收敛到方程的根，因此会导致后面的计算都会错下去。当然，可以在每步求解的时候根据 fsolve 返回的 exitflag（fsolve 函数的第三个返回参数）来判断是否接受当前解，如果不接受，再重新计算，但这样做势必增加计算量，而且操作不方便。

而用 ode15i 方法，给状态变量的一阶微分赋初值是个麻烦的事，若赋的值不好还经常导致"Convergence failure in DECIC."错误。在实际应用中，对一些较复杂的问题可以两种方法取长补短。例如利用 fsolve 函数求解状态变量的一阶微分初值值，然后将得到的初始值作为 decic 的 dy0 参数，进而得到 ode15i 所需要的状态变量一阶微分的初始值，这样发生"Convergence failure in DECIC."错误的概率会小些。example10_5_2 中第二种方法在给 dy0 赋初值时就参考了第一种方法 fsolve 给出的初始值。

10.5.3　延迟微分方程举例

DDE 是微分方程表达式要依赖某些状态变量过去一些时刻的状态，即形如：

$$y' = f(t, y, y(t-t_1), y(t-t_2), \cdots, y(t-t_n)) \tag{10-3}$$

其中，$t_1, t_2, \cdots, t_n > 0$，是时间延迟项，既可以是常数也可以是关于 t 和 y 的函数，当其是常数时

可以用 dde23 来求解,当是 t 和 y 的函数时可以用 ddesd 来求解。ddesd 也可以求解 $t_1,t_2,\cdots,t_n$ 为常数的情形,这时的用法和 dde23 类似。下面结合例子说明 dde23 和 ddesd 的用法。

【例 10.5-3】 求解下面的延迟微分方程:

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} 0.5y_3(t-3)+0.5y_2(t)\cos(t) \\ 0.3y_1(t-1)+0.7y_3(t)\sin(t) \\ y_2(t)+\cos(2t) \end{bmatrix}$$

当 $t\leqslant 0$ 时,$y_1(t)=1, y_2(t)=0, y_3(t)=1$。

dde23 的调用格式如下:

```
sol = dde23(ddefun,lags,history,tspan)
sol = dde23(ddefun,lags,history,tspan,options)
```

其中,ddefun 表示式(10-3)右边项的函数句柄,它代表的函数形式为:"dydt=ddefun (t,y,Z);",t 与当前的 t 对应,y 是一个列向量,是对 $y(t)$ 的近似。Z 的第 j 个列向量,即 Z(:,j)是对所有延迟为 t_j 的状态变量即 $y(t-t_j)$ 的估计,t_j=lags(j);lags 是存储各延迟常数的向量;history 是描述 $t\leqslant t0$ 时的状态变量的值的函数,可以为句柄形式或者常数形式;tspan 以及 options 的意思同其他 ode 求解函数;sol 为返回的求解结果,是一个结构体,其中 sol. x 是时间变量采样值,而 sol. y 为状态变量求解值。需要注意的是,这里的 sol. x 为行向量,sol. y 为行向量组成的矩阵,每一行表示一个求解出来的状态变量。

本例,方程自身已经是一阶微分方程组的标准形式了,故不用再另外转化。状态变量 $y_1(t)$ 和 $y_3(t)$ 分别存在时间为 1 和 3 的延迟。求解代码如下:

```
function example10_5_3
lags = [1,3]; % 延迟常数向量
history = [0,0,1]; % 小于初值时的历史函数
tspan = [0,8];
% 用 nested function 构造延迟微分方程组函数
    function dy = ddefunNestedFun(t,y,Z)
        y1d = Z(:,1); % 对所有延迟为 lags(1)的状态变量的近似
        y3d = Z(:,2); % 对所有延迟为 lags(2)的状态变量的近似
        % y3(t-3)的时间延迟了 lags(2),而 y3 又是第三个状态变量,因此 y3(t-3)用 y3d(3)来表示
        % 同理,y1(t-1)用 y1d(1)来表示。因此得到 dy 的如下表达式
        dy = [0.5 * y3d(3) + 0.5 * y(2) * cos(t);
            0.3 * y1d(1) + 0.7 * y(3) * sin(t);
            y(2) + cos(2 * t)];
    end
sol = dde23(@ddefunNestedFun,lags,history,tspan); % 调用 dde23 求解
% 以下画图呈现结果
plot(sol.x,sol.y(1,:),'k-','linewidth',2);
hold on
plot(sol.x,sol.y(2,:),'k-.','linewidth',2);
plot(sol.x,sol.y(3,:),'k-*','linewidth',1);
hold off
% 图例,自动选择最佳位置
L = legend('{\ity}_1(t)','{\ity}_2(t)','{\ity}_3(t)','Location','best');
```

```
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);title('方程各解的曲线图')
end
```

运行上述程序得到包含三个解的图形如图 10.13 所示。

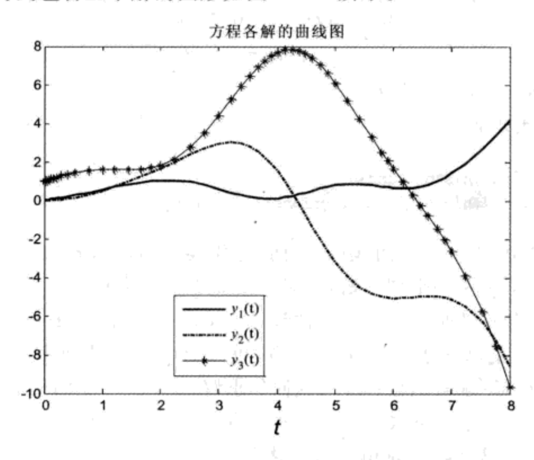


图 10.13　延迟为常数的 DDE 求解示例

dde23 只能求解延迟时间为常数的 DDE。下面再给出一个用 ddesd 求解延迟时间不为常数的 DDE 的例子。

【例 10.5-3】　求解如下延迟微分方程：

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -y_2(\exp(1-y_2(t)))y_2(t)^2\exp(1-y_2(t)) \end{bmatrix}$$

其中，求解时间范围为 tspan=[0.1,5]。

该延迟微分方程有解析解：$y_1(t)=\ln(t)$，$y_2(t)=\frac{1}{t}$，因此可以作为时间小于初值时的历史函数，同时也方便验证求解的结果。利用 ddesd 函数求解的代码如下：

```
function example10_5_4
    function v = ddex3hist(t)
        % 历史函数
        v = [log(t); 1./t];
    end
    function d = ddex3delay(t,y)
        % 延迟函数
        d = exp(1 - y(2));
    end
    function dydt = ddex3de(t,y,Z)
        % 延迟微分方程函数。由于只有一个延迟项，因此 Z 为 1 列的向量，y2(exp(1 - y2(t)))延迟
```

```
        % 了 exp(1 - y2(t)),而 y2 又是第二个状态变量,因此 y2(exp(1 - y2(t)))用 Z(2)来表示
        dydt = [ y(2); - Z(2) * y(2)^2 * exp(1 - y(2))];
    end
t0 = 0.1;
tfinal = 5;
tspan = [t0, tfinal]; % 求解时间范围
sol = ddesd(@ddex3de,@ddex3delay,@ddex3hist,tspan);
% 准确解
texact = linspace(t0,tfinal);
yexact = ddex3hist(texact);
% 以下画图呈现结果
figure;
plot(sol.x,sol.y(1,:),'o','markersize',7);
hold on
plot(sol.x,sol.y(2,:),'*','markersize',7);
plot(texact,yexact(1,:),'k-','linewidth',2);
plot(texact,yexact(2,:),'k:','linewidth',2);
% 图例,自动选择最佳位置
L = legend('{\ity}_1,ddesd','{\ity}_2,ddesd','{\ity}_1,解析解',...
    '{\ity}_2,解析解','Location','best');
set(L,'fontname','Times New Roman');
hold off
xlabel('\fontname{隶书}时间 t','fontsize',16);
ylabel('\fontname{隶书}y 的解','fontsize',16);
title('ddesd 求解和解析解对比图');
end
```

ddesd 求解的结果和解析解对比图形如图 10.14 所示。

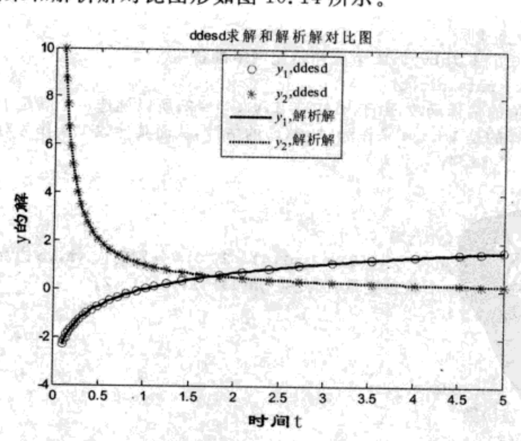


图 10.14 延迟不为常数的 DDE 求解示例

10.6 案例23：边值问题求解

10.6.1 概　述

前面讨论的 ode 系列函数只能用来求解初值问题，但是在实际中经常会遇到一些边值问题。例如热传导问题。初值时候的热源状态已知，一定时间后温度达到均匀。再如弦振动问题，弦两端端点的位置是固定的。像这种已知自变量在前后两端时系统状态的问题被称为边值问题，可用下面的方程来描述。

$$f(t,y,y')=0$$

定解条件：从 $y(0)=a, y(t_0)=b, y'(0)=c, y'(t_0)=d$ 中两端点 0 和 t_0 的两个表达式中各选一个组成定界条件。MATLAB 中提供了 bvp4c 和 bvp5c 函数用于求解边值问题。下面结合例子进行介绍。

10.6.2 求解案例

【例 10.6-1】 求解下列边值问题在区间 $t=[0,4]$ 上的解

$$y''=2y'\cos(t)-y\sin(4t)-\cos(3t)$$

边界条件为：$y(0)=1, y(4)=2$。

首先进行变量替换，化为标准形式。令 $y_1(t)=y(t), y_2(t)=y_1'(t)$，则上式可化为

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix}=\begin{bmatrix} y_2 \\ 2y_2\cos(t)-y_1\sin(4t)-\cos(3t) \end{bmatrix}$$

求解代码如下：

```
function example10_6_1
T = linspace(0,4,10); % 为 bvpinit 生成初始化网络准备
    function yinit = mat4init(t)
        % 对 y 初值的估计函数，由于 y1(0) = 1,y1(4) = 2;所以挑选一个满足上述条件的函数
        % 这里选择的是 1 + t/4 来作为对 y1(t)的估计，从而其导数 1/4 作为对 y2(t)的估计
        yinit = [  1 + t/4;
            1/4 ];
    end
    function res = mat4bc(ya,yb)
        % 边界条件：边界条件为 y1(0) = 1,y1(4) = 2; 0,4 分别对应 ya,yb 边界两端，而 y1,y1
        % 对应的都是第一个状态变量，因此是 ya(1) - 1,yb(1) - 2;
        res = [  ya(1) - 1
        yb(1) - 2];
    end
    function dydx = mat4ode(t,y)
        % 微分方程函数
        dydx = [ y(2)
            2 * y(2) * cos(t) - y(1) * sin(4 * t) - cos(3 * t) ];
    end
solinit = bvpinit(T,@mat4init); % 由 bvpinit 生成的初始化网格
```

```
sol = bvp4c(@mat4ode,@mat4bc,solinit); % 调用 bvp4c 求解,也可以换成 bvp5c
tint = linspace(0,4);
Stint = deval(sol,tint); % 根据得到的 sol 利用 deval 函数求出[0,4]区间内更多其他的解
% 画图展示
figure;
plot(tint,Stint(1,:),'k-','linewidth',2);
hold on
plot(tint,Stint(2,:),'k:','linewidth',2);
L = legend('{\ity}_1(t)','{\ity}_2(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);ylabel('方程的解');
title('求解结果')
end
```

运行上述代码,得到的结果如图 10.15 所示。

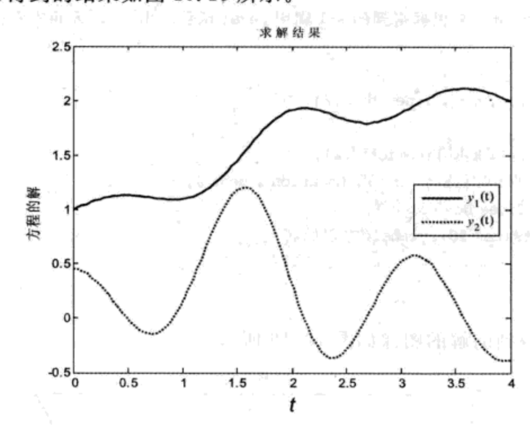


图 10.15　边值问题求解结果

看一个非线性边值的例子。

【例 10.6-2】 求解下列边值问题,求解范围 $t=[0,6]$:

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ \cos(y_2)\sin(y_1) \end{pmatrix}$$

边界条件为 $y_1(0)=1, y_1(6)=2$。该微分方程组第二个方程是关于状态变量的非线性函数,利用 bvp4c 的解法代码如下:

```
function example10_6_2
T = linspace(0,6,15); % 为 bvpinit 生成初始化网络准备
    function yinit = mat4init(t)
        % 对 y 初值的估计函数,由于 y1(0) = 1,y1(6) = 2;所以挑选一个满足上述条件的函数
        % 这里选择的是 1 + t/6 来作为对 y1(t)的估计,从而其导数 1/6 作为对 y2(t)的估计
        yinit = [  1 + t/6;
            1/6 ];
    end
```

```
    function res = mat4bc(ya,yb)
        % 边界条件;边界条件为 y1(0) = 1,y1(6) = 2; 0,6 分别对应 ya,yb 边界两端,而 y1,y1
        % 对应的都是第一个状态变量,因此是 ya(1) - 1,yb(1) - 2;
        res = [  ya(1) - 1
            yb(1) - 2];
    end
    function dydx = mat4ode(t,y)
        % 微分方程函数
        dydx = [ y(2)
            cos(y(2)) * sin(y(1)) ];
    end
solinit = bvpinit(T,@mat4init); % 由 bvpinit 生成的初始化网格
sol = bvp4c(@mat4ode,@mat4bc,solinit); % 调用 bvp4c 求解,也可以换成 bvp5c
tint = linspace(0,6);
Stint = deval(sol,tint); % 根据得到的 sol 利用 deval 函数求出[0,6]区间内更多其他的解
% 画图展示
figure;
plot(tint,Stint(1,:),'k-','linewidth',2);
hold on
plot(tint,Stint(2,:),'k:','linewidth',2);
L = legend('{\ity}_1(t)','{\ity}_2(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);ylabel('方程的解');
title('求解结果')
end
```

运行上述代码,得到的解的图像如图 10.16 所示。

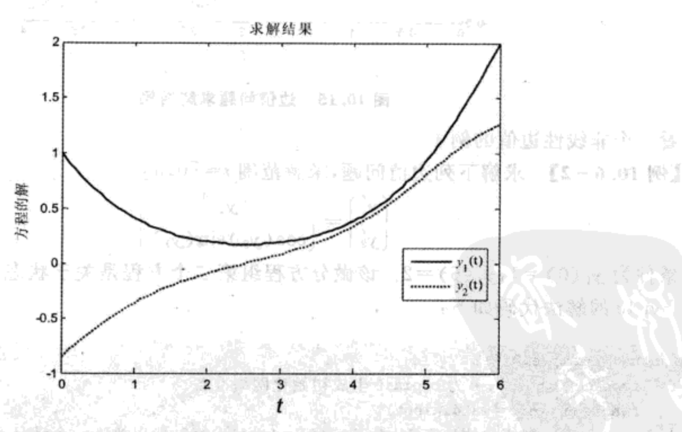


图 10.16　带参数边值问题的求解示例

再看一个带未知参数边值问题的求解例子。

【例 10.6-3】 求解下列边值问题(求解范围 $x=[0,\pi]$):

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2 \\ -(\lambda-2q\cos(2x))y_1 \end{bmatrix}$$

该方程的解 $y_1(x)$是 Mathieu 方程的特征函数，其中 $q=5$，λ 为未知参数。由于有 λ 这个未知参数，因此，方程有三个边界条件：$y_1(0)=1$，$y_2(0)=0$，$y_2(\pi)=0$。求解该问题的程序代码如下：

```
function example10_6_3
X = linspace(0,pi,10); % 为 bvpinit 生成初始化网络准备
q = 5;
lambda = 15; % 对 lambda 的猜测值
    function yinit = mat4init(x)
        % 对 y 初值的估计函数，由于 y2(0) = 1,y2(pi) = 0;所以挑选一个满足上述条件的函数
        % 这里选择的是 - k * sin(4 * x)来作为对 y2(t)的估计，其原函数 y1(t)满足 y1(0) = 1,因此
        % k = 4,所以 y1(t)的估计为 cos(4 * x),y2(t)的估计为 - 4 * sin(4 * x)
        yinit = [cos(4 * x)
            - 4 * sin(4 * x) ];
    end
    function res = mat4bc(ya,yb,lambda)
        % lambda 作为未知参数不能省略
        % 边界条件;边界条件为 y1(0) = 1,y2(0) = 2,y2(pi) = 0; 0,0,pi 分别对应 ya,ya,yb
        % 而 y1,y2,y2 分别是第一、第二、第二个状态变量。因此得到如下 res 表达式
        res = [ya(1) - 1;
            ya(2);
            yb(2)
            ];
    end
    function dydx = mat4ode(x,y,lambda)
        % 微分方程函数
        dydx = [ y(2);
            - (lambda - 2 * q * cos(2 * x)) * y(1) ];
    end
solinit = bvpinit(X,@mat4init,lambda); % 由 bvpinit 生成的初始化网格
sol = bvp4c(@mat4ode,@mat4bc,solinit); % 调用 bvp4c 求解,也可以换成 bvp5c
tint = linspace(0,pi);
Stint = deval(sol,tint); % 根据得到的 sol 利用 deval 函数求出[0,4]区间内更多其他的解
% 画图展示
figure;
plot(tint,Stint(1,:),'k-','linewidth',2);
hold on
plot(tint,Stint(2,:),'k:','linewidth',2);
L = legend('{\ity}_1(t)','{\ity}_2(t)','Location','best');
set(L,'fontname','Times New Roman');
xlabel('\itt','fontsize',16);ylabel('方程的解');
title('方程的解 y_1(t)为 Mathieu 方程的特征函数')
end
```

运行上述代码得到结果如图 10.17 所示。

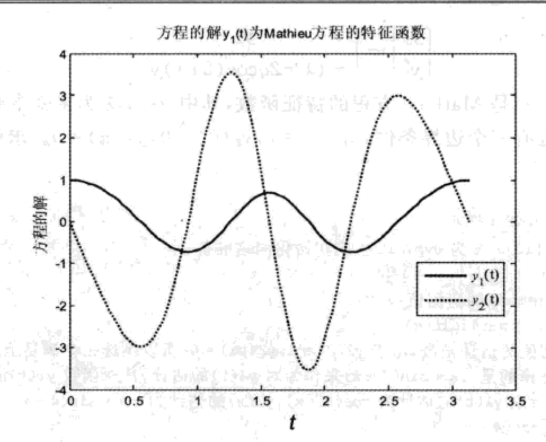


图 10.17　带未知参数边值问题的求解示例(Mathieu 方程的特征函数)

10.6.3　对 bvp4c 和 bvp5c 的改进

在 3.3.2 节曾经提到过在生成稀疏矩阵时，准备好数据利用 sparse 函数直接生成的效率要比先用 spalloc 函数创建一个稀疏矩阵 *S*，然后在程序中用“S(i,j)＝数据”赋值的方式高。仿真科技论坛的 junziyang 网友发现在 bvp4c 和 bvp5c 中大量使用后者的方式访问稀疏矩阵。因此他对上述两个函数做了相应的改进，发现改进后的程序在方程数较少的情况下，优化前后的差别还不明显，但随着方程数量的增加，效率提升越来越明显。图 10.18 所示是他试验的方程数为 156 个时，bvp4c 和 bvp5c 优化前后用 profiler 得到的计算时间开销对比。其中 bvp4c 和 bvp5c 是 MATLAB 的自带函数；bvp4c2 和 bvp5c2 是优化后的函数。实验环境：操作系统为 Windows XP sp3，CPU 为 Pentium D 915，内存为 1GB，MATLAB 版本为 R2009a。

Profile Summary
Generated 09-Jun-2009 15:41:47 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
bvp4c>colloc_Jac	30	857.583 s	834.654 s	
bvp4c	14	894.677 s	26.935 s	
funfun\private\odenumjac	949	22.883 s	11.603 s	
CWAmplifier>bvpfun	156900	10.120 s	7.538 s	
bvp4c2	14	71.247 s	26.831 s	
CW\private\odenumjac	949	22.579 s	11.803 s	
bvp4c2>colloc_Jac	30	34.324 s	7.971 s	~107×
CW\private\bvpcheckmatrix>condaux	130	7.465 s	7.465 s	
CWAmplifier>bvpfun	156900	9.883 s	7.162 s	
bvp5c>colloc_JacODE_region	23	3762.531 s	3749.489 s	
bvp5c	14	3940.027 s	126.792 s	
funfun\private\bvpcheckmatrix>condaux	138	26.187 s	26.187 s	
CWAmplifier	1	3983.965 s	13.048 s	
funfun\private\bvpcheckmatrix	23	40.533 s	12.471 s	
funfun\private\odenumjac	600	15.125 s	8.308 s	
bvp5c2	14	215.047 s	127.614 s	
CW\private\bvpcheckmatrix>condaux	132	25.731 s	25.731 s	
CWAmplifier	1	232.127 s	16.969 s	
CW\private\sparse2 (MEX-function)	23	13.191 s	13.191 s	
bvp5c2>colloc_JacODE_region	23	42.611 s	10.557 s	~355×
CW\private\bvpcheckmatrix	23	37.653 s	10.265 s	
CW\private\odenumjac	600	14.597 s	7.738 s	

图 10.18　改进前后的 bvp4c 和 bvp5c 函数效率对比

图 10.18 中画虚线框的部分是两个函数优化的那部分代码优化前后的运行时间。由此可见,对于一些较大型的问题,优化后的 sparse 矩阵处理部分的速度有 100 倍以上的提升。下面的链接里有 bvp4c2 和 bvp5c2 的代码。

bvp4c2 的链接:http://www.mathworks.com/matlabcentral/fileexchange/24359

或者:http://cid-909609e321d6e37a.skydrive.live.com/self.aspx/.Public/bvp4c2.zip

bvp5c2 的链接:http://www.mathworks.com/matlabcentral/fx_files/24360/1/bvp5c2.zip

或者:http://cid-909609e321d6e37a.skydrive.live.com/self.aspx/.Public/bvp5c2.zip

使用方法:将下载的文件解压到 MATLAB 安装路径下的 toolbox\matlab\funfun\文件夹下,在用户程序中将使用的 bvp4c/bvp5c 改为 bvp4c2/bvp5c2 即可。

第11章

案例24:层次分析法及其MATLAB实现

11.1 层次分析法概述

简单来说,层次分析法(Analytic Hierarchy Process,AHD)就是将要决策的问题以及和这些问题有关的因素分解成目标、准则、方案等层次,在此基础之上进行定性和定量分析的决策方法。该方法是美国运筹学家 T. L. Saaty 教授于 20 世纪 70 年代在为美国国防部研究"根据各个工业部门对国家福利的贡献大小而进行电力分配"课题时提出来的。层次分析法的主要特征是合理地将定性与定量的决策结合起来,按照思维、心理的规律把决策过程层次化、数量化。由于层次分析法在处理复杂的决策问题上的实用性和有效性,使得其提出后很快在世界范围得到重视,目前层次分析法应用已遍及经济、管理、能源政策制定和能源分配、行为科学、军事指挥、运输、农业、教育、人才、医疗和环境等各个领域。

层次分析法的基本思路与人们对一个决策问题的思维、判断过程大体上是一样的。以买笔记本电脑为例,假如有三种不同款式不同品牌的笔记本电脑 A,B,C 供选择,我们一般会根据价格、外观、重量、用途、功耗、品牌等一些准则去反复比较这三个候选。首先,会确定这些准则在自己心目中各占多大比重,不同的人这种比重会有很大差异。例如:喜欢玩一些大型 3D 游戏的人自然很看重其硬件性能;而只是用来上网聊天,躺在床上看电影的人则会优先考虑其重量以及散热量;预算有限的人还会对价格予以较多关注。其次,还会就每一个准则将 A,B,C 进行对比,比如 A 最便宜,B 次之;C 性能最好,B 次之;B 最轻巧,C 的品牌最知名等。最后,要将这两个层次的比较判断进行综合,在 A、B、C 中确定一台作为最符合自己需求的电脑。

11.2 层次分析法实现步骤

11.2.1 层次分析法的主要实现步骤

层次分析法在分析问题时主要分以下四步进行:

1) 分析系统中各个因素之间的关系,建立系统的递阶层次结构。

2) 对同一层次的各元素关于上一层次中某一准则的重要性进行两两比较,构造两两比较判断矩阵。

3) 由判断矩阵计算被比较元素对于该准则的相对权重以及一致性检验。

4) 计算各层元素对系统目标的合成权重,并进行排序。

下面对各步骤进行介绍。

11.2.2 建立层次分析的结构模型

用层次分析法分析问题,首先要做的是把问题条理化、层次化,构造层次分析的结构模型。



这些层次大体上可分为以下三类。

1) 最高层:在这一层次中只有一个元素,一般是分析问题的预定目标或理想结果,因此又称目标层。

2) 中间层。又称准则层,包括了为实现目标所涉及的中间环节。这一层可由若干层次组成,包括所需要考虑的准则、子准则。

3) 最底层:表示为实现目标可供选择的各种措施、决策、方案等,因此又称为措施层或方案层。

层次分析结构中的各项称为此结构模型中的元素。各层之间的关系如图 11.1 所示。

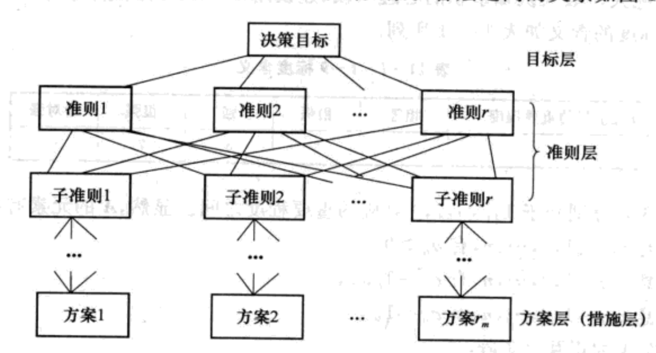


图 11.1　AHP 各层关系图

层次之间的支配关系不一定是完全的,即可以有元素(非底层元素)并不支配下一层次的所有元素而只支配其中部分元素。这种自上而下的支配关系所形成的层次结构,称为递阶层次结构。递阶层次结构中的层次数与问题的复杂程度及分析的详尽程度有关,所以,层次数一般可不受限制。

为了避免由于支配的元素过多而给两两比较判断带来困难,每层次中各元素所支配的元素一般不要超过九个,若多于九个时,可将该层次再划分为若干子层。

例如本章刚开始提到的买笔记本电脑模型,可以建立如图 11.2 所示的递阶层次结构。

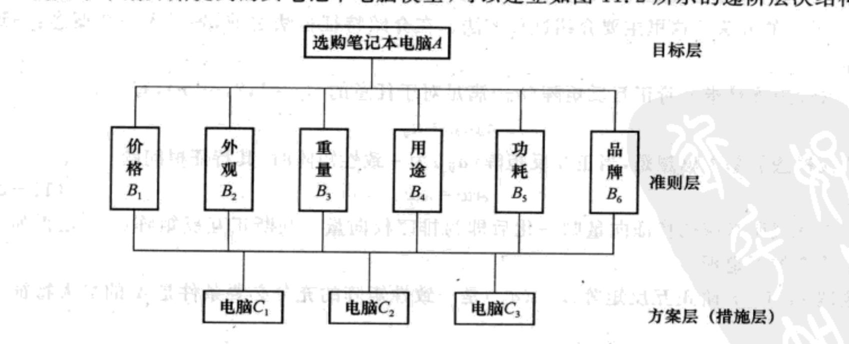


图 11.2　选购电脑层次结构模型图

11.2.3 构造成对比较矩阵

构造好层次分析的模型后，针对某一层来讲，在比较第 i 个元素与第 j 个元素相对上一层某个因素的重要性时，使用数量化的相对权重 a_{ij} 来表示，假设共有 n 个元素参与比较，则下列矩阵：

$$\mathbf{A}=(a_{ij})_{n\times n} \tag{11-1}$$

称为成对比较矩阵，也称成对判断矩阵。

Saaty 根据绝大多数人认知事物的心理习惯，建议用 1～9 及其倒数作为标度来确定 a_{ij} 的值。1～9 比例标度的含义如表 11－1 所列。

表 11－1　1～9 标度含义

i 比 j 强的重要程度	相等	稍强	强	很强	绝对强
a_{ij}	1	3	5	7	9

其中，2，4，6，8 分别介于 1，3，5，7，9 对应的重要程度之间。显然，$\mathbf{A}$ 的元素有如下特征。

1）对于任意 $i,j=1,2,\cdots,n$，有 $a_{ij}>0$。

2）对于任意 $i,j=1,2,\cdots,n$，有 $a_{ij}=1/a_{ij}$。

3）对于任意 $i,j=1,2,\cdots,n$，有 $a_{ii}=1$。

称成对比较矩阵 $\mathbf{A}$ 为正互反矩阵。

例如，对于上面选购笔记本电脑的模型来说，可以根据实际三台电脑的重量得到电脑对准则层 B_3 的正互反矩阵（a_{ij} 可以取笔记本电脑 j 和 i 的重量之比）：

$$\mathbf{A}_{B_3-C}=\begin{pmatrix}1 & 1/3 & 1/5\\ 3 & 1 & 3/5\\ 5 & 5/3 & 1\end{pmatrix}$$

11.2.4 单一准则下元素相对排序权重计算及比较矩阵一致性检验

单一准则下，由比较矩阵导出元素相对排序权重的方法有多种，其中特征根法是提出最早，应用最广的方法。这里主要介绍这种方法。在介绍特征根法之前，先引入一个概念：一致性矩阵。

一致性矩阵是指 n 阶正互反矩阵（a_{ij}）满足对于任意的 $i(j=1,2,\cdots,n)$，有

$$a_{ik}a_{kj}=a_{ij}$$

特征根法的基本思想是，当正互反矩阵（a_{ij}）为一致性矩阵时，其特征根问题

$$\mathbf{A}w=\lambda w \tag{11-2}$$

的最大特征根所对应的特征向量归一化后即为排序权向量。判断正互反矩阵（a_{ij}）是否为一致性矩阵有如下定理：

定理 11.1　n 阶正互反矩阵 $\mathbf{A}=(a_{ij})$ 是一致性矩阵的充分必要条件是 $\mathbf{A}$ 的最大特征值 $\lambda_{\max}=n$。

特征根法要求正互反矩阵（a_{ij}）为一致性矩阵。在实际操作中，由于客观事物的复杂性以及人们对事物判别比较时的模糊性，很难构造出完全一致的判断矩阵。因此 Satty 在构造层

次分析法时，提出满意一致性的概念，即用 **A** 的最大特征值 λ_{max} 与 n 的接近程度来作为一致性程度的尺度。这就涉及对得到的正互反矩阵 **A** 的一致性检验问题。其一致性检验的步骤如下：

1) 计算矩阵 **A** 的最大特征值 λ_{max}。

2) 求一致性指标(Consistency Index)

$$\text{C. I.} = \frac{\lambda_{max} - n}{n-1} \tag{11-3}$$

3) 查表求相应的平均随机一致性指标(Rondom Index)。平均随机一致性指标可以预先计算制成表。其计算过程如下：

取定阶数 n，随机取 9 标度数构造正互反矩阵后求其最大特征值，共计算 m 次(m 足够大)。计算这 m 个最大特征值的平均值 $\tilde{\lambda}_{max}$，得到：

$$\text{R. I.} = \frac{\tilde{\lambda}_{max} - n}{n-1} \tag{11-4}$$

Satty 以 $m=1000$ 得到表 11-2。

表 11-2　不同阶数矩阵的随机一致性指标

矩阵阶数	3	4	5	6	7	8
R. I.	0.58	0.90	1.12	1.24	1.32	1.41
矩阵阶数	9	10	11	12	13	
R. I.	1.45	1.49	1.51	1.54	1.56	

我们可以利用 MATLAB 实现 Satty 计算随机一致性指标的方法，当矩阵阶数大于 13 时，可以按照程序计算得到相应的 R. I.。实现上述方法的代码如下：

```
function RI = CalcRI(n)
% 对于给定的矩阵阶数 n,计算随机一致性指标 R.I.
if n == 1|| n == 2
    RI = 0;
    return
end
m = 1000; % 随机取 9 标度数构造正互反矩阵后求其最大特征值,计算 m 次
pool = [1:9,1./(2:9)]; % 9 标度所有可能的数字
lambda = zeros(m,1); % 存储每次得到的正互反矩阵的最大特征值
for k = 1:m
    ind = unidrnd(17,n,n); % 随机在 1～17 中生成 3 * 3 的矩阵,
    tmp1 = triu(pool(ind),1); % 提取 ind 的上三角矩阵,并以此为索引生成本次正互反矩阵的
    % 上三角
    tmp2 = triu(1./pool(ind),1); % 上三角取倒数,后面转置后作为随机正互反矩阵的下三角
    A = tmp1 + tmp2' + eye(n); % 生成随机正互反矩阵
    lambda(k) = max(eig(A));
end
RI = (mean(lambda) - n)/(n - 1);
```

运行上述代码得到如下结果：

```
N = 1:20;
RI = zeros(1,length(N));
for k = 3:length(N)
RI(k) = CalcRI(k);
end
RI
RI =
  Columns 1 through 10
    0   0  0.5678  0.9110  1.1161  1.2594  1.3393  1.4019  1.4427  1.4993
  Columns 11 through 20
    1.5103  1.5465  1.5607  1.5760  1.5773  1.6005  1.6002  1.6156  1.6224  1.6270
```

由此可见，程序计算出的结果和表 11-2 很接近，实际计算中由于随机数的产生不可能每次都一样，因此每次计算得到的 R. I. 都不尽相同，但范围都很接近。

4) 计算一致性比率 C. R. (Consistency Ratio)：

$$\text{C. R.} = \frac{\text{C. I.}}{\text{R. I.}}$$

5) 判断，当 C. R. <0.1 时，认为判断矩阵 $\boldsymbol{A}$ 有满意一致性；否则，若 C. R. ≥0.1，应考虑修正判断矩阵 $\boldsymbol{A}$。

11.2.5 各元素对目标层合成权重的计算过程

为了实现层次分析法的最终目的，需要从上而下逐层进行各层元素对目标合成权重的计算。

设已计算出第 $k-1$ 层 n_{k-1} 个元素相对于目标的合成权重为

$$\boldsymbol{w} = (w_1^{(k-1)}, w_2^{(k-1)}, \cdots, w_{n_k}^{(k-1)})^{\mathrm{T}} \tag{11-5}$$

再设第 k 层的 n_k 个元素关于第 $k-1$ 层第 j 个元素($j=1,2,\cdots,n_{k-1}$)的单一准则排序权重向量为

$$\boldsymbol{u}_j^{(k)} = (u_{1j}^{(k)}, u_{2j}^{(k)}, \cdots, u_{n_kj}^{(k)})^{\mathrm{T}} \tag{11-6}$$

式(11-6)应对第 k 层的 n_k 个元素是完全的。当某些元素不受 $k-1$ 层第 j 个元素支配时，相应位置用零补充，于是得到 $n_k \times n_{k-1}$ 矩阵：

$$\boldsymbol{U}^{(k)} = \begin{bmatrix} u_{11}^{(k)} & u_{12}^{(k)} & \cdots & u_{1n_{k-1}}^{(k)} \\ u_{21}^{(k)} & u_{22}^{(k)} & \cdots & u_{2n_{k-1}}^{(k)} \\ \vdots & \vdots & & \vdots \\ u_{n_k1}^{(k)} & u_{n_k2}^{(k)} & \cdots & u_{n_kn_{k-1}}^{(k)} \end{bmatrix} \tag{11-7}$$

利用式(11-5)和式(11-7)可得到第 k 层 n_k 个元素关于目标层的合成权重：

$$\boldsymbol{w}^{(k)} = \boldsymbol{U}^{(k)} \boldsymbol{w}^{(k-1)} \tag{11-8}$$

分解可得

$$\boldsymbol{w}^{(k)} = \boldsymbol{U}^{(k)} \boldsymbol{U}^{(k-1)} \cdots \boldsymbol{U}^{(3)} \boldsymbol{w}^{(2)} \tag{11-9}$$

将式(11-9)写成分量形式，有

$$w_i^{(k)} = \sum_{j=1}^{n_{k-1}} u_{ij}^{(k)} w_j^{(k-1)}, \quad i = 1,2,\cdots,n_k \tag{11-10}$$

各层元素对目标层的合成排序权重向量是否可以满意接受，同单一准则下的排序问题一样，需要进行综合一致性检验。

设 k 层的综合指标分别为一致性指标 C. I. $^{(k)}$，随机一致性指标 R. I. $^{(k)}$，一致性比率 C. R. $^{(k)}$。再设以第 $k-1$ 层上第 j 元素为准则的一致性指标为 C. I.$_j^{(k)}$，平均一致性指标为 R. I.$_j^{(k)}$ $(j=1,2,\cdots,n_{k-1})$，那么

$$\begin{aligned}\text{C. I.}^{(k)} &= (\text{C. I.}_1^k, \text{C. I.}_2^k, \cdots, \text{C. I.}_{n_{k-1}}^k)\, w^{(k-1)} \\ &= \sum_{j=1}^{n_{k-1}} w_j^{(k-1)} \text{C. I.}_j^k\end{aligned} \tag{11-11}$$

$$\begin{aligned}\text{R. I.}^{(k)} &= (\text{R. I.}_1^k, \text{R. I.}_2^k, \cdots, \text{R. I.}_{n_{k-1}}^k)\, w^{(k-1)} \\ &= \sum_{j=1}^{n_{k-1}} w_j^{(k-1)} \text{R. I.}_j^k\end{aligned} \tag{11-12}$$

利用式(11-11)和式(11-12)，可计算综合一致性比率

$$\text{C. R.}^{(k)} = \frac{\text{C. I.}^{(k)}}{\text{R. I.}^{(k)}} \tag{11-13}$$

当 C. R. $^{(k)}<0.1$ 时，认为递阶层次结构在第 k 层以上的判断具有整体满意的一致性。

在实际应用中，整体一致性检验常常不予进行，主要原因是对整体进行考虑是十分困难的；另一方面，若每个单一准则下的判断具有满意一致性，而整体达不到满意一致性时，调整起来非常困难。这个整体满意一致性的背景不如单一准则下的背景清晰，它的必要性也有待进一步研究。

根据上面的分析讨论可以得到下面实现一般层次分析法的MATLAB程序：

```
function [W,ahpResult] = ahp(C)
%层次分析法建模
%C:n*1的cell型数组,n+1等于建立的层次模型的层。C存储层次分析建模法所需要的输入信息,
%C{k}里面存储的也是一个cell数组,假设第k层有m_k个元素,从左到右依次编号1,2,…,m_k,
%C{k}{1,j}存储的是第k+1层各元素对第k层j元素的成对比较矩阵(j=1,2,…,m),C{k}{2,j}
%存储的是一m_k+1*1的逻辑数组,1表示第k+1层相应位置的元素受第k层j元素的支配,0表示不受
%支配。
%W:方案层对目标层的最终权重
%ahpResult:n*1的cell型数组。ahpResult存储层次分析建模法各层的结果信息,ahpResult{k}
%里面返回的也是一个cell数组,ahpResult{k}{1,j}返回的是k+1层所有元素相对k层j元素的
%权重,k+1层元素不受k层j元素支配的权重为0;ahpResult{k}{2,j}返回的是C{k}{1,j}中成对比
%较矩阵的最大特征值;ahpResult{k}{3,j}返回的是相应的一致性比率
RI=[0  0  0.58  0.90  1.12  1.24  1.32  1.41  1.45  1.49  1.51]; %平均随机一致性指标
n=length(C); %得到C的长度n,于是知道层数为n+1
ahpResult=cell(n,1); %存储各层结果信息
for k=1:n
    m_k=size(C{k},2); %k层的元素个数
    ahpResult{k}=cell(m_k,1);
    for kk=1:m_k
        %求第k+1层各元素对第k层kk元素的成对比较矩阵的特征值和特征向量
        [V,D]=eig(C{k}{1,kk});
        [maxD,ind]=max(diag(D)); %求最大特征值和其位置
```

```
        % 为存储第 k + 1 层所有元素相对 k 层 kk 元素的权重预留出空间，长度应等于 C{k}{2,kk}的长度
        ahpResult{k}{1,kk} = zeros(length(C{k}{2,kk}),1);
        % 将相应正互反矩阵属于最大特征值的特征向量归一化后赋给 ahpResult{k}{1,kk}中相应
        % 位置，这些位置由逻辑数组 C{k}{2,kk}决定
        ahpResult{k}{1,kk}(C{k}{2,kk}) = V(:,ind)/sum(V(:,ind));
        ahpResult{k}{2,kk} = maxD; % C{k}{1,kk}正互反矩阵的最大特征值
        nn = size(C{k}{1,kk},1); % C{k}{1,kk}的阶数
        ahpResult{k}{3,kk} = (maxD - nn)/(nn - 1)/RI(nn); % 相应的一致性比率
    end
end
W = ahpResult{1}{1,1};
for k = 2:n
    % cat(2,ahpResult{k}{1,:})把 k + 1 层所有元素相对 k 层各个元素的权重向量横向排在一起生成
    % 权重矩阵，即书中式(11 - 7)
    W = cat(2,ahpResult{k}{1,:}) * W;
end
```

11.3 应用实例

【例 11.3-1】 某工厂有一笔企业留成利润，需要决定如何分配使用。已经决定有三种用途：用做奖金、集体福利设施及引进技术设备。考察的准则也有三个：是否能调动职工的积极性、是否有利于提高技术水平及考虑改善职工生活条件。由此建立如图 11.3 所示的层次分析模型。

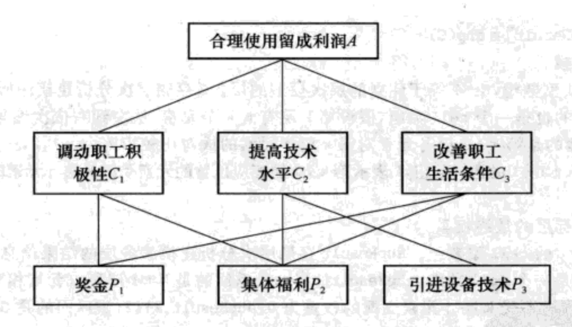


图 11.3 使用利润层次结构模型图

经过工厂决策人员讨论，可以得到如下各判断矩阵。

C 层关于目标层的判断矩阵为

A	C_1	C_2	C_3
C_1	1	1/5	1/3
C_2	5	1	3
C_3	3	1/3	1

由于 $C_1 \sim C_3$ 都受 A 的支配，因此相应的逻辑数组为[true true true]。

P 层对 C 层的判断矩阵如下：

P 层对 C_1 元素：

C_1	P_1	P_2
P_1	1	1/3
P_2	3	1

由于只有 P_1 和 P_2 受 C_1 的支配，因此相应的逻辑数组为[true　true　false]。

P 层对 C_2 元素：

C_2	P_1	P_2
P_1	1	1/5
P_2	5	1

由于只有 P_2 和 P_3 受 C_2 的支配，因此相应的逻辑数组为[false　true　true]。

P 层对 C_3 元素：

C_3	P_1	P_2
P_1	1	2
P_2	1/2	1

由于只有 P_1 和 P_2 受 C_3 的支配，因此相应的逻辑数组为[true　true　false]；

有了上述两两比较矩阵后，可以利用 ahp 函数来进行分析。构造 ahp 函数输入变量 C 以及求解过程如下：

```
clear
C = cell(2,1);
C{1}{1,1} = [1 1/5 1/3;5 1 3;3 1/3 1]; % 第二层(C层)关于第一层(目标层 A)的判断矩阵
C{1}{2,1} = true(3,1); % 相应的逻辑数组
C{2}{1,1} = [1 1/3;3 1]; % 第三层(P层)关于第二层 C1 的判断矩阵
C{2}{2,1} = [true true false]; % 相应的逻辑数组
C{2}{1,2} = [1 1/5;5 1]; % 第三层(P层)关于第二层 C2 的判断矩阵
C{2}{2,2} = [false true true]; % 相应的逻辑数组
C{2}{1,3} = [1 2;1/2 1]; % 第三层(P层)关于第二层 C3 的判断矩阵
C{2}{2,3} = [true true false]; % 相应的逻辑数组
[W,ahpResult] = ahp(C); % 调用 ahp 求解
>> W
W =
    0.1984
    0.2708
    0.5308
```

最后得到的 W 就是方案层各个方案所占的比重。由此可见，引进技术设备所占比重最大，改善员工福利次之。体现在资金分配上，即用全部留成利润的 53.08%引进技术设备，27.08%改善员工福利，剩下的 19.84%发奖金。

【例 11.3-2】 假设某人在制定食谱时有三类食品可选：肉、面包和蔬菜。这三类食品所含营养成分及单价如表 11-3 所列。

表 11-3 食品营养成分及单价

食 品	维生素 A/(IU·g)	维生素 B_2/(mg·g)	热量 Q/(kJ·g)	单价/(元·g)
肉	0.3527	0.0021	11.93	0.0275
面包	0.0005	0.0006	11.51	0.006
蔬菜	25.0	0.002	1.04	0.007

假设该人体重为 55kg，每天对各类营养的最小需求为

维生素 A　　7 500 国际单位(IU)

维生素 B_2　　1.6338 mg

热量(Q)　　8 548.5 kJ

问题是：应如何制定食谱使得在保证营养的前提下支出最小？

如果单纯考虑问题条件，容易建立下面的简单线性规划模型：

设选择肉为 x_1，面包为 x_2，蔬菜为 x_3，则有

$$\begin{cases} \min\ f=0.0275x_1+0.006x_2+0.007x_3 \\ \text{s.t.}\quad 0.3527x_1+0.0005x_2+25.0x_3\geqslant 7500 \\ \qquad 0.0021x_1+0.0006x_2+0.002x_3\geqslant 1.6338 \\ \qquad 11.93x_1+11.51x_2+1.04x_3\geqslant 8548.5 \\ \qquad x_1,x_2,x_3\geqslant 0 \end{cases} \tag{11-14}$$

对于上面的问题，利用 MATLAB 中求解线性规划的函数 linprog，很容易就求出最优解以及最优值，求解代码如下：

```
clear
f = [0.0275;0.006;0.007];
A = -[0.3527 0.0005 25;0.0021 0.0006 0.002;11.93 11.51 1.04];
b = -[7500;1.6338;8548.5];
options = optimset('LargeScale','off','Simplex','on');
[x,fval] = linprog(f,A,b,[],[],[0;0;0],[inf inf inf],[],options)
Optimization terminated.
x =
         0
  687.5267
  610.6420
fval =
    8.3997
```

求解出来的结果就是不吃肉，选择面包 687.44 g，蔬菜 610.67 g，每日最低支出为 8.40 元。读者到这里可能已经注意到，这个方案在实际中是很难让人接受的，因为它没有考虑到营养均衡，虽然总的营养够，但是食物不多样化，完全偏向价格便宜、营养相对丰富的食物，显得不"人性化"。解决这个问题的一个比较好的思路就是利用层次分析法。使用层次分析法可以将人们的偏好因素以及食物的均衡性引入到问题的求解中。对于上面的问题可以建立如

图 11.4所示的层次结构。

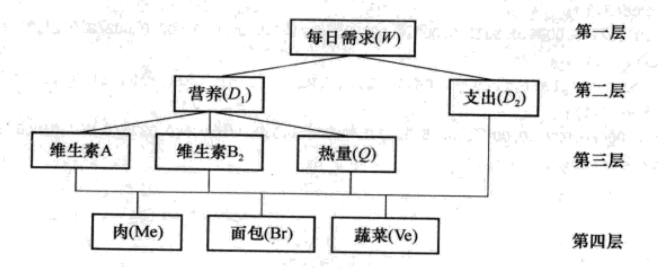


图 11.4 食谱营养分析层次结构图

有了上面的结构图,可以根据偏好建立两两比较矩阵,当然偏好因人而异。例如可以得到如下两两比较矩阵。

第二层对第一层的比较矩阵:C{1}{1,1}=[1 3;1/3 1],由于营养和支出都要考虑,所以相应的逻辑数组为 C{1}{2,1}=true(2,1)。

从图 11.4 看出第二层支出 D_2 直接支配第四层,因此,第三层第四个元素还是支出 D_2,这样 C{2}{1,1}=[1 1 2;1 1 2; 1/2 1/2 1](三种营养元素对营养的判断矩阵);相应的逻辑矩阵为 C{2}{2,1}=[true true true false];

C{2}{1,2}=1(支出对支出的判断矩阵);相应的逻辑矩阵为 C{2}{2,2}=[false,false,false,true];

第三层对第四层元素的两两比较矩阵可以用原始的营养成分以及单价的数据得到,即

C{3}{1,1}=[1,0.3527/0.0005,0.3527/25;0.0005/0.3527,1,0.0005/25;25/0.3527,25/0.0005,1];(三种食品对维生素 A 的判断矩阵);相应的逻辑数组:C{3}{2,1}=true(3,1);

C{3}{1,2}=[1,0.0021/0.0006,0.0021/0.002;0.0006/0.0021,1,0.0006/0.002;0.002/0.0021,25/0.0006,1];(三种食品对维生素 B_2 的判断矩阵);相应的逻辑数组:C{3}{2,2}=true(3,1);

C{3}{1,3}=[1,11.93/11.51,11.93/1.04;11.51/11.93,1,11.51/1.04;1.04/11.93,1.04/11.51,1];(三种食品对热量的判断矩阵);相应的逻辑数组:C{3}{2,3}=true(3,1);

C{3}{1,4}=[1,0.006/0.0275,0.007/0.0275;0.0275/0.006,1,0.007/0.006;0.0275/0.007,0.006/0.007,1](三种食品对单价的判断矩阵,注意单价越高越不重要);相应的逻辑数组:C{3}{2,4}=true(3,1)。

至此,层次分析法函数 ahp 的输入参数 C 都有了,可以运行下面的代码:

```
clear
C = cell(3,1);
C{1}{1,1} = [1 3;1/3 1];
C{1}{2,1} = true(2,1);
C{2}{1,1} = [1 1 2;1 1 2; 1/2 1/2 1];
C{2}{2,1} = [true true true false];
C{2}{1,2} = 1;
C{2}{2,2} = [false,false,false,true];
```

```
C{3}{1,1} = [1,0.3527/0.0005,0.3527/25;0.0005/0.3527,1,0.0005/25;25/0.3527,25/0.0005,1 ];
C{3}{2,1} = true(3,1);
C{3}{1,2} = [1,0.0021/0.0006,0.0021/0.002;0.0006/0.0021,1,0.0006/0.002;0.002/0.0021,25/0.0006,1 ];
C{3}{2,2} = true(3,1);
C{3}{1,3} = [1,11.93/11.51,11.93/1.04;11.51/11.93,1,11.51/1.04;1.04/11.93,1.04/11.51,1 ];
C{3}{2,3} = true(3,1);
C{3}{1,4} = [1,0.006/0.0275,0.007/0.0275;0.0275/0.006,1,0.007/0.006;0.0275/0.007,0.006/0.007,1 ];
C{3}{2,4} = true(3,1);
[W,ahpResult] = ahp(C);
W
W =
    0.1064
    0.1918
    0.7019
```

上述结果表明，按这个人的情况，肉、面包和蔬菜的比例取 0.1064，0.1918，0.7019 比较合适，引入参变量 k，令 $x_1=0.1064k$，$x_2=0.1918k$，$x_3=0.7019k$，将其带入式(11-14)则得到新的线性规划模型：

$$\begin{cases} \min f=0.009k \\ \text{s.t. } 17.5851k \geqslant 7500 \\ \quad 0.0017k \geqslant 1.6338 \\ \quad 4.2069k \geqslant 8548.5 \\ \quad k \geqslant 0 \end{cases} \tag{11-15}$$

手算即可求解，得 $k=2\,032$，因此得到 $x=(216.2, 389.7, 1\,426.3)$；即每日食用肉 216.2 g，面包 389.7 g，蔬菜 1 426.3 g。每日食品支出为 18.29 元。

当然对于不同的人可以有不同的判断矩阵，如果认为食品支出过高，可以修改第二层对第一层的比较矩阵，如修改成“[1 1/3;3 1];”，类似的计算可得到：

$$x=(145.0, 529.2, 699.7)$$

即每日食用肉 145.0 克，面包 529.2 克，蔬菜 699.7 克。每日食品支出为 12.06 元。

第 12 章

案例 25：定时器及其应用

12.1 定时器介绍

12.1.1 概 述

从 MATLAB 6.5 版本开始，MATLAB 开始支持定时器。相对于传统的 pause 函数，定时器要强大得多，不仅可以等效实现 pause 的功能，还可以实现更多 pause 无法实现的功能，一些需要在一定周期内独立运行的程序特别适合利用定时器来实现。

完整建立一个定时器，并让它发挥作用，一直到最后删除需要以下几个步骤：第一，利用 timer 函数建立定时器；第二，利用 start 函数启动定时器；第三，利用 stop 函数停止定时器；第四，利用 delete 函数删除定时器；第五，利用 clear 函数删除代表定时器句柄的变量。

定时器的建立非常类似于 GUI 中控件的建立，即都有专门的函数来建立，并可以返回代表其本身的句柄，并可以在需要时利用 set 函数操作其句柄来完成属性的修改和设置。下面着重介绍定时器的属性。

12.1.2 定时器属性介绍

定时器支持的属性，即 timer 或者 set 函数支持的属性见表 12－1。

表 12－1 定时器属性表

属性名称	属性描述	数据类型、值、默认值、访问情况
AveragePeriod	定时器启动后，执行的所有 TimerFcn 之间的平均间隔时间	double 型数值； 默认为空；只有“ExecutionMode”设置为除“singleShot”外的其他值，并且 TimerFcn 实际被执行两次及两次以上时才有确定的值； 始终是只读型的
BusyMode	当上一次的 timerfcn 未执行完，而新的 timerfcn 又要被执行，定时器所采取的操作。有三种不同操作： “drop”(不执行 timerfcn 函数)； “error”(产生错误，此种情况下要求 ErrorFcn 事先设置好，并执行 ErrorFcn)； “queue”：timerfcn 进入队列，并在下一次可执行的时候执行	枚举字符串类型，值有“drop”，“error”，“queue”三种类型； 默认为“drop”； 当属性“Running”为“on”时是只读的，即此时不可修改“BusyMode”属性

续表 12－1

属性名称	属性描述	数据类型、值、默认值、访问情况
ErrorFcn	当遇到错误时，定时器所执行的函数。该函数要在 StopFcn 之前执行	ErrorFcn 类型：字符串，函数句柄或者 cell 数组； 默认为空； 一直是可以改写的
ExecutionMode	定时器的执行模式	类型：枚举字符串； 可取的值： （"singleShot"、"fixedDelay"、"fixedRate"、"fixedspacing"）； 默认："singleShot"； 当属性"Running"设为"on"时只读
InstantPeriod	最后执行的两个 TimerFcn 之间的时间间隔	double 型数值； 默认为 NaN； 只支持只读
Name	用户自己设定的定时器的名字	字符串类型； 默认"timer－i"，i 表示按先后顺序设置的第 i 个定时器，执行"delete(timerfindall)"命令可以使得 i 的计数重新从 1 开始； 始终是只读型的
ObjectVisibility	给开发者提供一种途径用来阻止终端用户访问定时器。如果某定时器的"ObjectVisibility"属性设为"off"，则该定时器对 timerfind 函数不可见 该属性不影响定时器的其他性质，开发者可以在定时器创立的 m 文件中设置该属性	类型：枚举字符串； 可取的值：（"off"，"on"）； 默认的值："on"； 一直是可以改写的
Period	TimerFcn 的执行周期，单位是秒	任何大于 0.001 的 double 型数值； 默认值是 1.0； 当属性"Running"设为 on 时只读
Running	表明当前定时器是否正在执行	类型：枚举字符串； 值："off"，"on"； 默认值："off"； 始终是只读型的
StartDelay	指定定时器从开始运行到第一次执行 TimerFcn 之间的时间延迟，单位是秒	任何大于等于 0 的 double 型数值； 默认值是 0； 当属性"Running"设为"on"时只读
StartFcn	定时器启动的时候执行的函数	StartFcn 类型：字符串，函数句柄或者 cell 数组； 默认为空； 一直是可以改写的

续表 12－1

属性名称	属性描述	数据类型、值、默认值、访问情况
StopFcn	定时器停止的时候执行的函数。定时器停止的情况有以下几种： 1）调用 stop 函数停止定时器的执行； 2）定时器完成了 TasksToExecute 设定的执行次数，即每执行一次 TimerFcn，TasksExecuted 增加一次，当其达到 TasksToExecute 设定的执行次数时，定时器停止； 3）错误发生导致定时器停止时，这时先执行 ErrorFcn 再执行 StopFcn	StopFcn 类型：字符串，函数句柄或者 cell 数组； 默认为空； 一直是可以改写的
Tag	用户自己给定时器设定的标签	字符串类型； 默认是空字符串(' ')； 一直是可以改写的
TasksToExecute	指定定时器需要执行 TimerFcn 函数的次数	任何大于 0 的 double 型数值； 默认值为 1； 一直是可以改写的
TasksExecuted	从定时器开始运行时起，记录 TimerFcn 调用的次数，TimerFcn 每执行一次，TasksExecuted 自动加 1	任何大于等于 0 的 double 型数值； 默认值为 0； 始终是只读型的
TimerFcn	定时器的回调函数	TimerFcn 类型：字符串，函数句柄或者 cell 数组； 默认为空； 一直是可以改写的
Type	标明定时器的类型	字符串类型； 仅取一个值："timer"； 始终是只读的
UserData	用户存储的一些数据	类型：用户指定的类型； 默认值：空； 一直是可以改写的

表 12－1 列出了定时器支持的属性，下面对其中部分属性进行补充说明。

ErrorFcn，StartFcn，StopFcn 以及 TimerFcn：这些属性是定时器用以实现操作功能的重要组成部分，四个属性对应着定时器不同时刻不同状态下执行的函数，关于函数的设定，这四个属性支持的格式完全一样，即可以是字符串、函数句柄或者 cell 数组，默认都是空的，并且可以随时访问。下面通过一个简单的例子来说明。

【例 12.1－1】 运行下列代码，体会定时器中各类型回调函数的用法：

```
function example12_1_1
% 定义定时器 TimerA
TimerA = timer('StartFcn','disp(''定时器启动！'')','ErrorFcn',@ErrorFun,...
    'TimerFcn',{@TimerFun,rand},'stopfcn',{@stopfun,'定时器停止！'},...
    'executionmode','fixedspacing','period',2,'TasksToExecute',10);
% 定时器的 ErrorFun
```

```
    function ErrorFun(object1,event1)
        if isequal(object1,TimerA) % 判断 ErrorFun 默认的第一个参数是不是对应的 Timer 对象
            disp('ErrorFun 默认的第一个输入参数为其对应的 timer 对象');
            % 默认的第二个参数 event1 存储了错误发生的一些信息，如下展示
            disp('ErrorFun 默认的第二个输入参数 event1 信息如下:')
            disp(['event1.Type = ',event1.Type])
            disp(['event1.Data.message = ',event1.Data.message])
            disp(['event1.Data.messageID = ',event1.Data.messageID])
            disp(['event1.Data.time = ',mat2str(event1.Data.time)])
        else
            disp('错误发生！')
        end
    end
% 定时器的 TimerFun
    function TimerFun(object2,event2,a)
        disp(['a = ',num2str(a)]);
        if get(TimerA,'TasksExecuted') == 5
            set(TimerA,'TasksExecuted',5);
        end
    end
% 定时器的 stopfun
    function stopfun(object2,event2,str)
        disp(str);
    end
start(TimerA) % 启动定时器
wait(TimerA); % 等待定时器执行完毕
end
```

运行上述程序结果如下：

```
定时器启动！
a = 0.2785
a = 0.2785
a = 0.2785
a = 0.2785
a = 0.2785
??? Error while evaluating TimerFcn for timer 'timer - 1'

Changing the 'TasksExecuted' property of timer objects is not allowed.

ErrorFun 默认的第一个输入参数为其对应的 timer 对象
ErrorFun 默认的第二个输入参数 event1 信息如下：
event1.Type = ErrorFcn
event1.Data.message = Changing the 'TasksExecuted' property of timer objects is
not allowed.
event1.Data.messageID = MATLAB:class:SetDenied
event1.Data.time = [2010 3 3 22 52 25.265]
定时器停止！
```

上述程序定义了一个定时器 TimerA，TimerA 的 StartFcn 采用的是字符串类型，ErrorF-

cn 采用的是函数句柄，而 TimerFcn 和 StopFcn 由于有额外参数输入，因此采用的是 cell 数组。这四种函数如果自己写独立的函数进行定义时，所定义函数都默认必须有两个参数，第一个参数是对应的定时器句柄，第二个参数是事件信息，包含了触发这个函数时的一些时间、消息等信息，第三个参数开始才是用户传入的参数，这点和 GUI 设计中控件的回调函数一样。上述程序中，TimerFun 传入的参数是 rand，stopfun 传入的参数是"定时器停止！"这个字符串。需要说明的是，定时器建立起来，TimerFun 等函数的输入参数也就定了，以后也不会改变了，除非重新设置。这就是为什么上述代码运行结果中，a 一直等于 0.278 5 不变。

TimerA 的执行模式是 fixedspacing（几种模式的区别稍后介绍），周期为 2 s，"TasksToExecute"设为 10 表示不发生错误的前提下循环 10 次定时器执行完毕。

利用 start 函数启动定时器，利用 wait 函数等待定时器执行完毕，定时器启动后首先执行 StartFcn，即显示"定时器启动！"，之后开始执行 TimerFun，TimerFun 内部首先显示用户的输入参数 a 等于多少，并判断已经执行的 TimerFun 次数是否等于 5，当等于 5 的时候再设置一遍已经执行的 TimerFun 次数，注意到"TasksExecuted"的属性始终是只读型的，因此这时候会发生错误，ErrorFun 被执行。从 ErrorFun 的执行可以清楚看到，ErrorFun 第一个默认的参数就是对应的定时器，第二个参数记录了发生错误的一些信息。当错误发生时，首先执行 ErrorFun，然后定时器停止，这时才执行 stopfun。

BusyMode 和 ExecutionMode

【例 12.1-2】 下列程序显示 BusyMode 的性质。

```
function example12_1_2
    function TimerFun(object,event)
        t = clock; % 函数刚开始执行时间点
        pause(1); % 函数执行了 1 s
        et = etime(t,get(object,'userdata')); % 函数开始运行和上次函数开始运行时间间隔
        disp(['本次函数开始执行距上次函数开始执行时间为:',num2str(et),' s! '])
        set(object,'userdata',t); % 记录本次函数开始运行时间
    end
% 定义定时器 TimerA, 'executionmode'设为'fixedSpacing'
TimerA = timer('StartFcn','disp(''定时器 A 开始:'')','ErrorFcn',...
    'disp(''A 发生错误'')','StartDelay',0.5,'TimerFcn',@TimerFun,'stopfcn',...
    'disp(''定时器 A 停止！ '')','executionmode','fixedrate','period',0.5,...
    'TasksToExecute',3,'busymode','error');
set(TimerA,'userdata',clock); % 先给 userdata 赋值
start(TimerA);
wait(TimerA);
set(TimerA,'userdata',clock,'busymode','drop');
start(TimerA);
wait(TimerA);
end
```

运行结果如下：

```
定时器 A 开始:
本次函数开始执行距上次函数开始执行时间为:0.5 s!
定时器 A 停止!
```

```
A 发生错误
定时器 A 开始：
本次函数开始执行距上次函数开始执行时间为：0.5 s!
本次函数开始执行距上次函数开始执行时间为：1 s!
本次函数开始执行距上次函数开始执行时间为：1 s!
定时器 A 停止！
```

当 BusyMode 设为"error"时候，由于 TimerFun 函数执行的时间长于 period 的 0.5 s（TimerFun 内部有一个暂停 1 s），因此下一个 TimerFun 开始执行的时候，上一个还没有执行完，这时发生错误，从而显示"A 发生错误"。而把 BusyMode 设为"drop"或者"queue"时则没有上述错误。

ExecutionMode（执行的模式），四种模式的意义如下：

"singleShot"模式：只能执行一次，其他模式都可以执行多次。

"fixedSpacing"模式：上一次执行完毕的时刻到下一次被加入队列的时刻之间的间隔是指定的固定时间长度。

"fixedDelay"模式：上一次开始执行的时刻到下一次被加入队列的时刻之间的间隔是指定的固定时间长度。

"fixedRate"模式：两次被加入到执行语句队列的时刻之间的间隔是指定的固定时间长度。

在 MATLAB 的帮助文档中搜索"Differences - Between - Execution - Modes"可以看到关于上述四种模式的直观的说明图。下面的代码演示了"fixedSpacing"和"fixedDelay"以及"fixedRate"的异同。

【例 12.1-3】 运行下列代码，体会 ExecutionMode 不同模式的区别。

```
function example12_1_3
% 定时器的 TimerFun，供 TimerA,TimerB,TimerC 共用
    function TimerFun(object,event)
        t = clock; % 函数刚开始执行时间点
        pause(1); % 函数执行了 1 s
        et = etime(t,get(object,'userdata')); % 函数开始运行和上次函数开始运行时间间隔
        disp(['本次函数开始执行距上次函数开始执行时间为：',num2str(et),' S! '])
        set(object,'userdata',t); % 记录本次函数开始运行时间
    end
% 定义定时器 TimerA,'executionmode'设为'fixedSpacing'
TimerA = timer('StartFcn','disp(''定时器 A 开始：'')','ErrorFcn',...
    'disp(''A 发生错误'')','StartDelay',0.5,'TimerFcn',@TimerFun,'stopfcn',...
    'disp(''定时器 A 停止！ '')','executionmode','fixedSpacing','period',0.5,...
    'TasksToExecute',3);
% 定义定时器 TimerB,'executionmode'设为'fixedDelay'
TimerB = timer('StartFcn','disp(''定时器 B 开始：'')','ErrorFcn',...
    'disp(''B 发生错误'')','StartDelay',0.5,'TimerFcn',@TimerFun,'stopfcn',...
    'disp(''定时器 B 停止！ '')','executionmode','fixedDelay','period',0.5,...
    'TasksToExecute',3);
% 定义定时器 TimerC,'executionmode'设为'fixedRate'
TimerC = timer('StartFcn','disp(''定时器 C 开始：'')','ErrorFcn',...
    'disp(''C 发生错误'')','StartDelay',0.5,'TimerFcn',@TimerFun,'stopfcn',...
    'disp(''定时器 C 停止！ '')','executionmode','fixedRate','period',0.5,...
```

```
        'TasksToExecute',3);
    % 运行定时器 A
    set(TimerA,'userdata',clock);
    start(TimerA);
    wait(TimerA);
    % 运行定时器 B
    set(TimerB,'userdata',clock);
    start(TimerB);
    wait(TimerB);
    % 运行定时器 C
    set(TimerC,'userdata',clock);
    start(TimerC);
    wait(TimerC);
    end
```

运行结果如下：

```
定时器 A 开始：
本次函数开始执行距上次函数开始执行时间为:0.5 S!
本次函数开始执行距上次函数开始执行时间为:1.5 S!
本次函数开始执行距上次函数开始执行时间为:1.5 S!
定时器 A 停止！
定时器 B 开始：
本次函数开始执行距上次函数开始执行时间为:0.5 S!
本次函数开始执行距上次函数开始执行时间为:1 S!
本次函数开始执行距上次函数开始执行时间为:1 S!
定时器 B 停止！
定时器 C 开始：
本次函数开始执行距上次函数开始执行时间为:0.5 S!
本次函数开始执行距上次函数开始执行时间为:1 S!
本次函数开始执行距上次函数开始执行时间为:1 S!
定时器 C 停止！
```

TimerA，TimerB 和 TimerC 的区别仅仅在于"executionmode"，从上面的结果可以看出，"executionmode"设为"fixedSpacing"时，相邻两个 TimerFun 之间的执行时间为 period＋函数体执行时间。而设为"fixedRate"和"fixedDelay"时，由于队列延迟时间相对于函数体本身执行的时间来说很小，因此可以近似认为相邻两个 TimerFun 之间的执行时间为 period 的时间。

12.2　定时器应用举例

本小节以两个实例来说明定时器的一些用途，希望能给读者一些启发。

【例 12.2－1】　利用定时器实现每隔随机的一段时间（$0<t<5$ s）显示"Hello World"的程序。

通过前面的讨论，我们知道可以通过定时器实现每隔一定时间进行特定的操作，其实，利用定时器的"StartDelay"属性，还可以实现每隔随机的一段时间进行特定的操作。下面的程序以最简单的显示"Hello World"为例，说明了如何实现每隔随机的一段时间显示"Hello World"。

```
function example12_2_1
TimerA = timer('timerfcn',@TimerFun,'StartDelay',unidrnd(5000,1)/1000,...
    'executionmode','fixedrate');
 % 用来记录 TimerFun 函数执行的次数，由于通过 StartDelay 方式实现每隔随机一段时间进行特定
 % 操作时，需要反复停止与启动定时器，因此 TasksExecuted 就不准确了。
set(TimerA,'userdata',0);
num = 5; % 随机显示 5 次后，就停止定时器
start(TimerA)
    function TimerFun(obj,event)
        set(TimerA,'userdata',get(TimerA,'userdata') + 1);
        disp(['间隔时间为 ',num2str(get(TimerA,'startDelay')),' S! ']);
        disp('Hello World! ');
        stop(TimerA);
        set(TimerA,'StartDelay',unidrnd(5000,1)/1000);
        start(TimerA);
        while get(TimerA,'userdata') == num
            stop(TimerA)
            return;
        end
    end
end
```

运行结果如下：

```
间隔时间为 0.71 S!
Hello World!
间隔时间为 2.109 S!
Hello World!
间隔时间为 4.579 S!
Hello World!
间隔时间为 3.962 S!
Hello World!
间隔时间为 4.798 S!
Hello World!
```

【例 12.2-2】 利用定时器模拟下面的过程：圆形跑道周长是 500 m，甲、乙两人按顺时针沿圆形跑道同时、同地起跑，甲每分钟跑 60 m，乙每分钟跑 50 m，甲、乙两人每跑 200 m 均要停下来休息 1 min，问：甲首次套圈乙用多长时间？

如果用代数思路的话我们可以这样分析这个问题：

以 200 m 为一个周期，甲跑 200 m 需 200/60＝10/3 min，加上休息的 1 min，一个周期是 13/3 min。同理可求出乙的周期是 5 min，他们周期的最小公倍数是 65 min，也就是说 65 min 后，甲乙休息完毕又同时起跑。可以求出，65 min 甲完成了 15 个周期，即 3 000 m，乙完成了 13 个周期，即 2 600 m，甲超过乙 400 m。甲再追赶乙 100 m 即可。让我们看看再过 10 min 后，乙这时候刚好又完成两个周期，共跑了 3 000 m。准备新的周期，甲也完成两个周期，并且第三个周期跑了 4/3 min，也就是说又追了乙 80 m。这时候离甲休息还有 10/3－4/3＝2 min，每分钟甲追乙 10 m，所以甲要休息时刚好追上乙。总共用时 77 min。

下面的代码是模拟真实情况的动画，利用定时器来控制小球运动。以 1 s 代表 1 min。定时器的周期设置为 0.01 s。

```
function AchaseB
r = 500/(2 * pi); % 圆形跑道半径
t = linspace(0,2 * pi,1000); % 一圈等分成 1000 份
RestAngle = 200/r; % 开始跑到休息跑过的角度
figure('menubar','none','toolbar','none','defaultuicontrolunits','normalized');
plot(r * cos(t),r * sin(t)); % 画跑道
hold on
% 甲的初始位置
A = plot(r * cos(0),r * sin(0),'marker','h','MarkerSize',8,'MarkerFaceColor','r');
aa = text(r * cos(0),r * sin(0),'甲');
% 乙的初始位置
B = plot(r * cos(0),r * sin(0),'marker','h','MarkerSize',8,'MarkerFaceColor','g');
bb = text(r * cos(0),r * sin(0),'乙');
% 建立控制甲乙位置的定时器 timerA,timerB
timerA = timer('TimerFcn',@TimerFcnA,'executionmode','fixedspacing','period',0.01);
timerB = timer('TimerFcn',@TimerFcnB,'executionmode','fixedspacing','period',0.01);
hold off
axis equal
axis off
start(timerA);start(timerB);

%%
    function TimerFcnA(timerA,event)
        nA = get(timerA,'TasksExecuted');
        % 为了方便较短时间内模拟追赶结果,以实际 1 s 代表甲乙追赶过程中的 1 min,甲跑 200 m
        % 用时 10/3 min,对应着 3.33 s,由于定时器 period 为 0.01 s,因此对应着
        % TasksExecuted 为 333,同理休息的 1 min 对应着 1 s,对应着 TasksExecuted 为 100
        % 这样根据 timerA 的 TasksExecuted,即 nA,对(333 + 100)一个跑步加休息周期,求余
        % 结果如果小于 333,则是在跑的过程,需要修改 A 的位置。否则是在休息过程,位置不变
        modnA433 = mod(nA,433);
        if modnA433 <= 333
            set(A,'xdata',r * cos( RestAngle * (floor(nA/433) + modnA433/333) ),...
                'ydata',r * sin( RestAngle * (floor(nA/433) + modnA433/333) ));
            set(aa,'position',[r * cos( RestAngle * (floor(nA/433) + modnA433/333) ),...
                r * sin( RestAngle * (floor(nA/433) + modnA433/333) )]);
            drawnow
        end
        Axdata = get(A,'xdata');
        Aydata = get(A,'ydata');
        Bxdata = get(B,'xdata');
        Bydata = get(B,'ydata');
        % 判断甲追上乙的条件。nA 大于 500 是满足套圈的必要条件,&& 右面的是保证甲乙之间距离
        % 足够小
        if nA > 500 && (abs(Axdata - Bxdata) + abs(Aydata - Bydata))< 1
            stop(timerA);
            stop(timerB);
            title(['甲追上乙的时间:',num2str(nA/100),'分']);
        end
    end
%%
```

```
    function TimerFcnB(timerB,event)
        nB = get(timerB,'TasksExecuted');
        %类似 TimerFcnA 中注释,判断 B 的位置
        modnB500 = mod(nB,500);
        if modnB500 <= 400
            set(B,'xdata',r * cos( RestAngle * (floor(nB/500) + modnB500/400) ),...
                'ydata',r * sin( RestAngle * (floor(nB/500) + modnB500/400) ));
            set(bb,'position',[r * cos( RestAngle * (floor(nB/500) + modnB500/400) ),...
                r * sin( RestAngle * (floor(nB/500) + modnB500/400) )])
            drawnow
        end
    end
end
```

图 12.1、图 12.2 分别是追赶过程中以及追上后的图片。

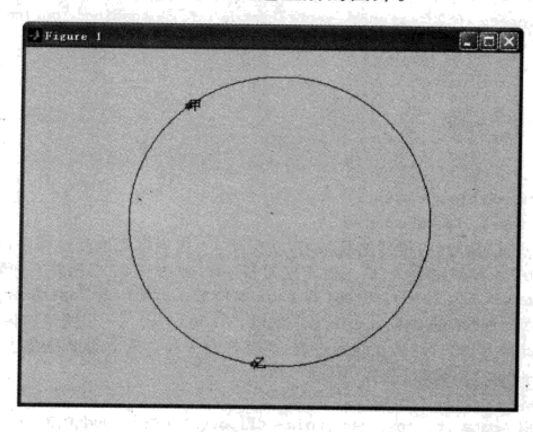


图 12.1　甲追乙过程中图片

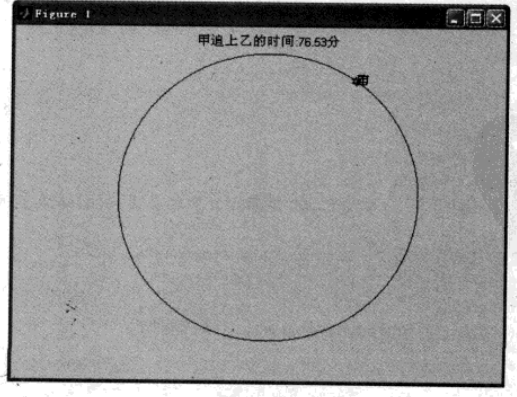


图 12.2　甲最后追上乙的图片

由于 10/3 min 应该精确对应 10/3 s，而程序是以 3.33 s 代替的 10/3 s，因此，追赶时间会和 77 s 有误差，不会严格相等。

作为示例程序，上述程序提供了一种利用定时器做动画的途径。实际中，一些需要独立控制的界面元素非常适合利用定时器来控制，充分利用定时器的功能可以设计出强大，形象直观的动画。

参考文献

[1] 姚海臣. 长输油管道控制系统资源配置优化研究[D]. 大庆:大庆石油学院,2007.

[2] 阎长罡,曹战,刘清荣. 基于遗传算法的车间设备布局设计研究[J]. 大连交通大学学报,2007,28(3):33～37.

[3] 曹战. 基于遗传算法的车间设备布局问题研究[D]. 大连:大连交通大学,2007.

[4] 卢开澄,卢华明. 线性规划[M]. 北京:清华大学出版社,2009.

[5] 黄红选,韩继业. 数学规划[M]. 北京:清华大学出版社,2006.

[6] 刘思峰,党耀国,方志耕,等. 灰色系统理论及其应用[M]. 3 版. 北京:科学出版,2004.

[7] 胡振涛,刘先省. 一种实用的数据融合算法[J]. 自动化仪表,2005,26(8):7～9.

[8] 王道林. 泰安市 GDP 预测的灰色模型[J]. 山东科技大学学报(自然科学版),2005,(2):77～79

[9] 范金城,梅长林. 数据分析[M]. 北京:科学出版社,2002.

[10] 邓聚龙. 灰色控制系统[M]. 2 版. 武汉:华中理工大学出版社,1997.

[11] 高惠璇. 应用多元统计分析[M]. 北京:北京大学出版社,2005.

[12] 刘正君. MATLAB 科学计算与可视化仿真宝典[M]. 北京:电子工业出版社,2009.

[13] 吴祈宗. 运筹学与最优化方法[M]. 北京:机械工业出版社,2003.

[14] 李庆扬,关治,白峰杉. 数值计算原理[M]. 北京:清华大学出版社,2000.

[15] 李星. 积分方程[M]. 北京:科学出版社,2008.

[16] Geoffrion A M, Graves G W. Multicommodity distribution system design by benders decomposition[J]. Management Science, 1974, 20(5):822～844

[17] Yang J, Zhang D, Frangi A, et al. Two-dimensional PCA: a new approach to appearance-based face representation and recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence [J], 2004, 26(1):131～137.

[18] Chen S, Zhu Y, Zhang D, et al. Feature extraction approaches based on matrix pattern: MatPCA and MatFLDA[J]. Pattern Recognition Letters, 2005, 26(8): 1157～1167.

[19] Zhang D, Zhou Z H. (2D)2PCA: two-directional two-dimensional PCA for efficient face representation and recognition [J]. Neurocomputing, 2005, 69(1～3):224～231.

[20] Rajkiran Gottumukkal, Vijayan K Asari. An improved face recognition technique based on modular PCA approach [J]. Pattern Recognition Letters , 2004, 25 (4): 429～436.

[21] Yang J, Yang J Y. From image vector to matrix: a straightforward image projection technique—IMPCA vs. PCA [J]. Pattern Recognition , 2002, 35(9):1997～1999.

[22] Won Young Yang, Wenwu Cao, Tae-Sang Chung, et. Applied numerical

methods using MATLAB[M], New Jersey, Hoboken:John Wiley & Sons, Inc, 2005.

[23] L. F. Shampine, M. W. Reichelt, J. A. Kierzenka. Solving index-1 DAEs in Matlab and Simulink[J]. SIAM Review, 1999,41: 538-552.

[24] Shampine L F, Gladwell I, Thompson S. Solving ODEs with MATLAB[M]. New York:Cambridge University Press,2003.

[25] Davis P J. Interpolation and Approximation[M]. Boston:Ginn(Blaisdell),Massachusetts,1963.